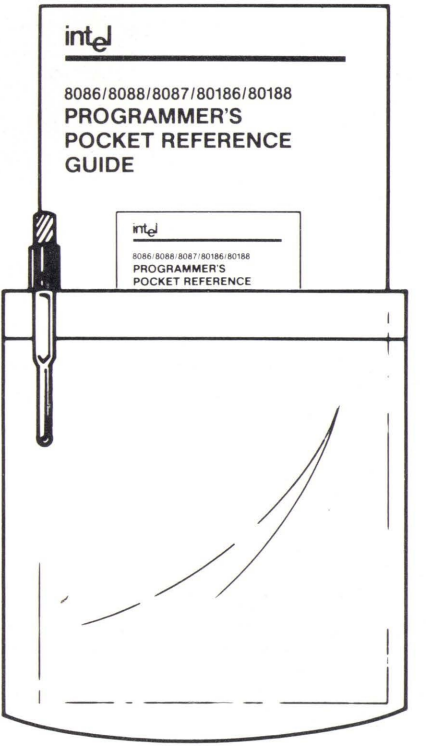
intel

8086/8088/8087/80186/80188 PROGRAMMER'S POCKET REFERENCE GUIDE

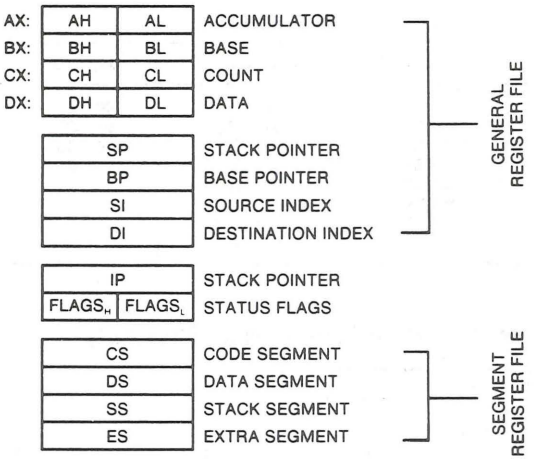


Order Number: 231017-001

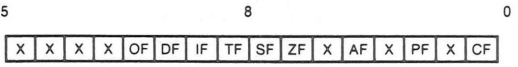
8086/8088/8087/80186/80188 PROGRAMMER'S POCKET REFERENCE GUIDE

CONTENTS

8086 Register Model



Instructions that reference the flag register file as a 16-bit object use the symbol FLAGS to represent the file:



X = Don't Care

Flags

AF: AUXILIARY CARRY — BCD
CF: CARRY FLAG
DF: DIRECTION FLAG (STRINGS)
IF: INTERRUPT ENABLE FLAG
OF: OVERFLOW FLAG (CF SF)
PF: PARITY FLAG
SF: SIGN FLAG
TF: TRAP (SINGLE STEP FLAG)
ZF: ZERO FLAG

Operand Summary

"reg" field Bit Assignments:

Word Operand	Byte Operand	Segment
000 AX	000 AL	00 ES
001 CX	001 CL	01 CS
010 DX	010 DL	10 SS
011 BX	011 BL	11 DS
100 SP	100 AH	
101 BP	101 CH	
110 SI	110 DH	
111 DI	111 BH	

Second Instruction Byte Summary

mod xxx r/m

mod	Displacement
00	DISP = 0*, disp-low and disp-high are absent
01	DISP = disp-low sign-extended to 16-bits, disp-high is absent
10	DISP = disp-high: disp-low
11	r/m is treated as a "reg" field

r/m	Operand Address
000	(BX) + (SI) + DISP
001	(BX) + (DI) + DISP
010	(BP) + (SI) + DISP
011	(BP) + (DI) + DISP
100	(SI) + DISP
101	(DI) + DISP
110	(BP) + DISP*
111	(BX) + DISP

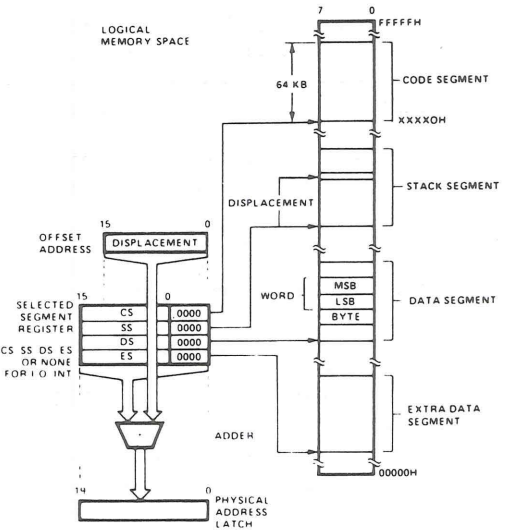
DISP follows 2nd byte of instruction (before data if required).

*except if mod = 00 and r/m = 110 then EA = disp-high: disp-low.

Operand Address (EA) Timing (Clocks):

Add 4 clocks for word operands at ODD ADDRESSES.
Immed Offset = 6
Base (BX, BP, SI, DI) = 5
Base + DISP = 9
Base + Index (BP + DI, BX + SI) = 7
Base + Index (BP + SI, BX + DI) = 8
Base + Index (BP + DI, BX + SI) + DISP = 11
Base + Index (BP + SI, BX + DI) + DISP = 12

Memory Segmentation Model



Segment Override Prefix

0 0 1 reg 1 1 0

Timing: 2 clocks

Use of Segment Override

Operand Register	Default	With Override Prefix
IP (code address)	CS	Never
SP (stack address)	SS	Never
BP (stack address or stack marker)	SS	BP + DS or ES, or CS
SI or DI (not incl. strings)	DS	ES, SS, or CS
SI (implicit source addr for strings)	DS	ES, SS, or CS
DI (implicit dest addr for strings)	ES	Never

8086/8088 Instructions

Notes for 8086/8088 Instructions

The individual instruction descriptions are shown by a format box such as the following:

These are byte-wise representations of the object code generated by the assembler and are interpreted as follows:

- Opcode is the 8-bit opcode for the instruction. The actual opcode generated is defined in the "Opcode" column of the instruction table that follows each format box.
- m/op/r/m is the byte that specifies the operands of the instruction. It contains a 2-bit mode field (m), a 3-bit register field (op), and a 3-bit register or memory (r/m) field.
- Dashed blank boxes following the m/op/r/m box are for any displacement required by the mode field.
- Data is for a byte of immediate data.
- A dashed blank box following a Data box is used whenever the immediate operand is a word quantity.

AAA = ASCII Adjust for Addition

Opcode

Opcode	Clocks	Operation
37	4	adjust AL, flags, AH

AAD = ASCII Adjust for Division

Long——Opcode

Opcode	Clocks	Operation
D5,0A	60	Adjust AL, AH prior to division

AAM = ASCII Adjust for Multiplication

Long——Opcode

Opcode	Clocks	Operation
D4,0A	83	Adjust AL, AH after multiplication

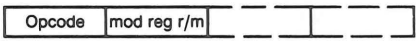
AAS = ASCII Adjust for Subtraction

Opcode

Opcode	Clocks	Operation
3F	4	adjust AL, flags, AH

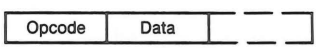
ADC = Integer Add with Carry

Memory/Reg + Reg



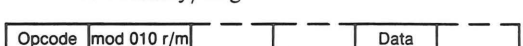
	Opcode	Clocks	Operation
Byte	12	3	Reg8 ← CF + Reg 8 + Reg8
	12	9+EA	Reg8 ← CF + Reg8 + Mem8
	10	16+EA	Mem8 ← CF + Mem8 + Reg8
Word	13	3	Reg16 ← CF + Reg16 + Reg16
	13	9+EA	Reg16 ← CF + Reg16 + Mem16
	11	16+EA	Mem16 ← CF + Mem16 + Reg16

Immed to AX/AL



	Opcode	Clocks	Operation
Byte	14	4	AL ← CF + AL + Immed8
Word	15	4	AX ← CF + AX + Immed16

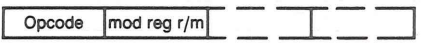
Immed to Memory/Reg



	Opcode	Clocks	Operation
Byte	80	4	Reg8 ← CF + Reg8 + Immed8
	80	17+EA	Mem8 ← CF + Mem8 + Immed8
Word	81	4	Reg16 ← CF + Reg16 + Immed16
	81	17+EA	Mem16 ← CF + Mem16 + Immed16
	83	4	Reg16 ← CF + Reg16 + Immed8
	83	17+EA	Mem16 ← CF + Mem16 + Immed8

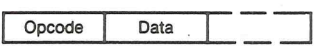
ADD = Integer Addition

Memory/Reg + Reg



	Opcode	Clocks	Operation
Byte	02	3	Reg8 ← Reg8 + Reg8
	02	9+EA	Reg8 ← Reg8 + Mem8
	00	16+EA	Mem8 ← Mem8 + Reg8
Word	03	3	Reg16 ← Reg16 + Reg16
	03	9+EA	Reg16 ← Reg16 + Mem16
	01	16+EA	Mem16 ← Mem16 + Reg16

Immed to AX/AL



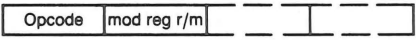
Opcode	Clocks	Operation
04	4	AL ← AL + Immed8
05	4	AX ← AX + Immed16

Immed to Memory/Reg

	Opcode	Clocks	Operation
Byte	80	4	Reg8 ← Reg8 + Immed8
	80	17+EA	Mem8 ← Mem8 + Immed8
Word	81	4	Reg16 ← Reg16 + Immed16
	81	17+EA	Mem16 ← Mem16 + Immed16
	83	4	Reg16 ← Reg16 + Immed8
	83	17+EA	Mem16 ← Mem16 + Immed8

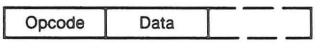
AND = Logical AND

Memory/Reg with Reg



	Opcode	Clocks	Operation
Byte	22	3	Reg8 ← Reg8 AND Reg8
	22	9+EA	Reg8 ← Reg8 AND Mem8
	20	16+EA	Mem8 ← Mem8 AND Reg8
Word	23	3	Reg16 ← Reg16 AND Reg16
	23	9+EA	Reg16 ← Reg16 AND Mem16
	21	16+EA	Mem16 ← Mem16 AND Reg16

Immed to AX/AL



	Opcode	Clocks	Operation
Byte	24	4	AL ← AL AND Immed8
Word	25	4	AX ← AX AND Immed16

Immed to Memory/Reg

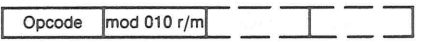
	Opcode	Clocks	Operation
Byte	80	4	Reg8 ← Reg8 AND Immed8
	80	17+EA	Mem8 ← Mem8 AND Immed8
Word	81	4	Reg16 ← Reg16 AND Immed16
	81	17+EA	Mem16 ← Mem16 AND Immed16

CALL = Call

Within segment or group, IP relative

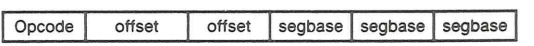
Opcode	Clocks	Operation
E8	19	IP ← IP + Disp16—(SP) ← return link

Within segment or group, Indirect



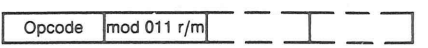
Opcode	Clocks	Operation
FF	16	IP ← Reg16—(SP) ← return link
FF	21 + EA	IP ← Mem16—(SP) ← return link
FF	21 + EA	IP ← Mem16—(SP) ← return link

Inter-segment or group, Direct



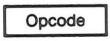
Opcode	Clocks	Operation
9A	28	CS ← segbase IP ← offset

Inter-segment or group, Indirect



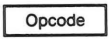
Opcode	Clocks	Operation
FF	37 + EA	CS ← segbase IP ← offset

CBW = Convert Byte to Word



Opcode	Clocks	Operation
98	2	convert byte in AL to word in AX

CLC = Clear Carry Flag



Opcode	Clocks	Operation
F8	2	clear the carry flag

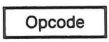
CLD = Clear Direction Flag

Opcode	Clocks	Operation
FC	2	clear direction flag

CLI = Clear Interrupt Enable Flag

Opcode	Clocks	Operation
FA	2	clear interrupt flag

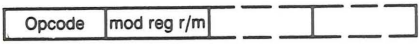
CMC = Complement Carry Flag



Opcode	Clocks	Operation
F5	2	complement carry flag

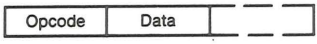
CMP = Compare Two Operands

Memory/Reg with Reg



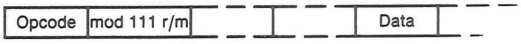
	Opcode	Clocks	Operation
Byte	38	3	flags ← Reg8 - Reg8
	38	9+EA	flags ← Reg8 - Mem8
	3A	9+EA	flags ← Mem8 - Reg8
Word	39	3	flags ← Reg16 - Reg16
	39	9+EA	flags ← Reg16 - Mem16
	3B	9+EA	flags ← Mem16 - Reg16

Immed to AX/AL



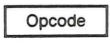
	Opcode	Clocks	Operation
Byte	3C	4	flags AL - Immed8
Word	3D	4	flags AX - Immed16

Immed to Memory/Reg



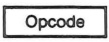
	Opcode	Clocks	Operation
Byte	80	4	flags ← Reg8 - Immed8
	80	10+EA	flags ← Mem8 - Immed8
Word	81	4	flags ← Reg16 - Immed16
	81	10+EA	flags ← Mem16 - Immed16
	83	4	flags ← Reg16 - Immed8
	83	10+EA	flags ← Mem16 - Immed8

CWD = Convert Word to Doubleword



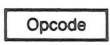
Opcode	Clocks	Operation
99	5	convert word in AX to doubleword in DX:AX

DAA = Decimal Adjust for Addition



Opcode	Clocks	Operation
27	4	adjust AL, flags, AH

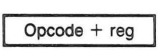
DAS = Decimal Adjust for Subtraction



Opcode	Clocks	Operation
2F	4	adjust AL, flags, AH

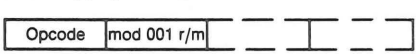
DEC = Decrement by 1

Word Register



Opcode	Clocks	Operation
48+reg	2	Reg16 ← Reg16 - 1

Memory/Byte Register



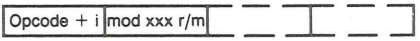
	Opcode	Clocks	Operation
Byte	FE	3	Reg8 ← Reg8 - 1
	FE	15+EA	Mem8 ← Mem8 - 1
Word	FF	15+EA	Mem16 ← Mem16 - 1

DIV = Unsigned Division

Memory/Reg with AX or DX:AX

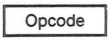
	Opcode	Clocks	Operation
Byte	F6	80-90	AH,AL ← AX / Reg8
	F6	(86-96)+EA	AH,AL ← AX / Mem8
Word	F7	144-162	DX,AX ← DX:AX / Reg16
	F7	(150-168)+EA	DX,AX ← DX:AX / Mem16

ESC = Escape



Opcode	Clocks	Operation
D8+i	8+EA	data bus ← (EA)
D8+i	2	data bus ← (EA)

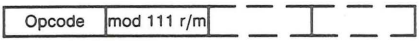
HLT = Halt



Opcode	Clocks	Operation
F4	2	halt operation

IDIV = Signed Division

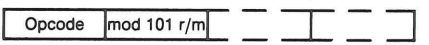
Memory/Reg with AX or DX:AX



	Opcode	Clocks	Operation
Byte	F6	101-112	AH,AL ← AX / Reg8
	F6	(107-118)+EA	AH,AL ← AX / Mem8
Word	F7	165-184	DX,AX ← DX:AX / Reg16
	F7	(171-190)+EA	DX,AX ← DX:AX / Mem16

IMUL = Signed Multiplication

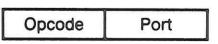
Memory/Reg with AL or AX



	Opcode	Clocks	Operation
Byte	F6	80-98	AX ← AL*Reg8
	F6	(86-104)+EA	AX ← AL*Mem8
Word	F7	128-154	DX:AX ← AX*Reg16
	F7	(134-160)+EA	DX:AX ← AX*Mem16

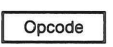
IN = Input Byte, Word

Fixed port



	Opcode	Clocks	Operation
Byte	E4	10	AL ← Port8
	E5	10	AX ← Port8

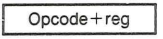
Variable port



	Opcode	Clocks	Operation
Word	EC	8	AL ← Port16(in DX)
	ED	8	AX ← Port16(in DX)

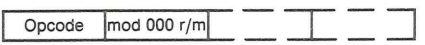
INC = Increment by 1

Word Register



Opcode	Clocks	Operation
40+reg	2	Reg16 ← Reg16 + 1

Memory/Byte Register

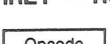


	Opcode	Clocks	Operation
Byte	FE	3	Reg8 ← Reg8 + 1
	FE	15+EA	Mem8 ← Mem8 + 1
Word	FF	15+EA	Mem16 ← Mem16 + 1

INT
INTO = Interrupt

Opcode	Clocks	Operation
CC	52	Interrupt 3
CD	51	Interrupt 'type'
CE	53 or 4	Interrupt4 if FLAGS.OF=1, else NOP

IRET = Return from Interrupt



Opcode	Clocks	Operation
CF	24	Return from interrupt

Jcond = Jump on Condition

Operation

```
if condition is true then do;
   sign-extend displacement to 16 bits;
   IP←IP + sign-extended displacement;
   end if;
```

Format

Opcode	Disp

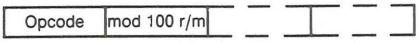
Opcode	Clocks	Operation	*cond* =
77	16 or 4	jump if above	JA
73	16 or 4	jump if above or equal	JAE
72	16 or 4	jump if below	JB
76	16 or 4	jump if below or equal	JBE
72	16 or 4	jump if carry set	JC
74	16 or 4	jump if equal	JE
7F	16 or 4	jump if greater	JG
7D	16 or 4	jump if greater or equal	JGE
7C	16 or 4	jump if less	JL
7E	16 or 4	jump if less or equal	JLE
76	16 or 4	jump if not above	JNA
72	16 or 4	jump if neither above nor equal	JNAE
73	16 or 4	jump if not below	JNB
77	16 or 4	jump if neither below nor equal	JNBE
73	16 or 4	jump if no carry	JNC
75	16 or 4	jump if not equal	JNE
7E	16 or 4	jump if not greater	JNG
7C	16 or 4	jump if neither greater nor equal	JNGE
7D	16 or 4	jump if not less	JNL
7F	16 or 4	jump if neither less nor equal	JNLE
71	16 or 4	jump if no overflow	JNO
7B	16 or 4	jump if no parity	JNP
79	16 or 4	jump if positive	JNS
75	16 or 4	jump if not zero	JNZ
70	16 or 4	jump if overflow	JO
7A	16 or 4	jump if parity	JP
7A	16 or 4	jump if parity even	JPE
7B	16 or 4	jump if parity odd	JPO
78	16 or 4	jump if sign	JS
74	18 or 6	jump if zero	JZ
E3	18 or 6	jump if CX is zero (does not test flags)	JCXZ

JMP = Jump

Within segment or group, IP relative

Opcode	Clocks	Operation
E9	15	IP ← IP + Disp16
EB	15	IP ← IP + Disp8 (Disp8 sign-extended)

Within segment or group, Indirect

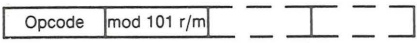


Opcode	Clocks	Operation
FF	11	IP ← Reg16
FF	18+EA	IP ← Mem16
FF	18+EA	IP ← Mem16

Inter-segment or group, Direct

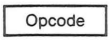
Opcode	Clocks	Operation
EA	15	CS ← segbase IP ← offset

Inter-segment or group, Indirect



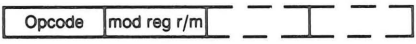
Opcode	Clocks	Operation
FF	24+EA	CS ← segbase IP ← offset

LAHF = Load AH from Flags



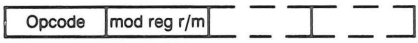
Opcode	Clocks	Operation
9F	4	copy low byte of flags word to AH

LDS/LES = Load Pointer to DS/ES and Register



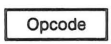
Opcode	Clocks	Operation
C4	16+EA	dword pointer at EA goes to reg16 (1st word) and ES (2nd word)
C5	16+EA	dword pointer at EA goes to reg16 (1st word) and DS (2nd word)

LEA = Load Effective Address



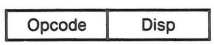
Opcode	Clocks	Operation
8D	2+EA	Reg16 ← EA

LOCK = Assert Bus Lock



Opcode	Clocks	Operation
F0	2	assert the bus lock next instruction

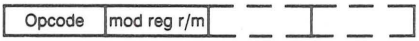
LOOP*xx* = Loop Control



Opcode	Clocks	Operation	*xx* =
E1	18 or 6	dec CX; loop if equal and CX not 0	LOOPE
E0	19 or 5	dec CX; loop if not equal and CX not 0	LOOPNE
E1	18 or 6	dec CX; loop if zero and CX not 0	LOOPZ
E0	19 or 5	dec CX; loop if not zero and CX not 0	LOOPNZ
E2	17 or 5	dec CX; loop if CX not 0	LOOP

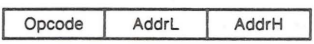
MOV = Move Data

Memory/Reg to or from Reg



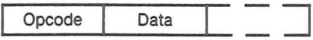
	Opcode	Clocks	Operation
Byte	88	9+EA	Mem8 ← Reg8
	88	2	Reg8 ← Reg8
	8A	8+EA	Reg8 ← Mem8
Word	89	9+EA	Mem16 ← Reg16
	89	2	Reg16 ← Reg16
	8B	8+EA	Reg16 ← Mem16

Direct-Addressed Memory to or from AX/AL



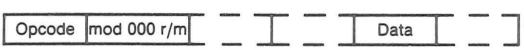
	Opcode	Clocks	Operation
Byte	A0	10	AL ← Mem8
	A2	10	Mem8 ← AL
Word	A1	10	AX ← Mem16
	A3	10	Mem16 ← AX

Immed to Reg



	Opcode	Clocks	Operation
Byte	B0+reg	4	Reg 8 ← Immed8
Word	B8+reg	4	Reg16 ← Immed16

Immed to Memory/Reg



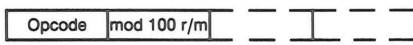
Opcode	Clocks	Operation
C6	4	Reg8 ← Immed8
C6	10+EA	Mem8 ← Immed8
C7	4	Reg16 ← Immed16
C7	10+EA	Mem16 ← Immed16

Memory/Reg to or from SReg

	Opcode	Clocks	Operation
Word	8C	9+EA	Mem16 ← SReg
	8C	2	Reg16 ← SReg
	8E	8+EA	SReg ← Mem16
	8E	2	SReg ← Reg16

MUL = Unsigned Multiplication

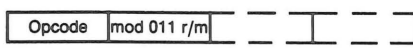
Memory/Reg with AL or AX



	Opcode	Clocks	Operation
Byte	F6	70-77	AX ← AL*Reg8
	F6	(76-83)+EA	AX ← AL*Mem8
Word	F7	118-133	DX:AX ← AX*Reg16
	F7	(124-139)+EA	DX:AX ← AX*Mem16

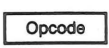
NEG = Negate an Integer

Memory/Reg



Opcode	Clocks	Operation
F6	3	Reg8 ← 00H - Reg 8
F7	3	Reg16 ← 0000H - Reg16
F6	16+EA	Mem8 ← 00H - Mem8
F7	16+EA	Mem16 ← 0000H - Mem16

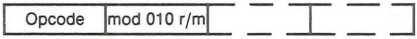
NOP = No Operation



Opcode	Clocks	Operation
90	3	no operation

NOT = Form One's Complement

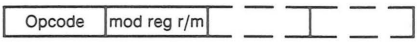
Memory/Reg



	Opcode	Clocks	Operation
Byte	F6	3	Reg8 ← 0FFH - Reg8
	F6	16+EA	Mem8 ← 0FFH - Mem8
Word	F7	3	Reg16 ← 0FFFFH - Reg16
	F7	16+EA	Mem16 ← 0FFFFH - Mem16

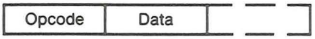
OR = Logical Inclusive OR

Memory/Reg with Reg



	Opcode	Clocks	Operation
Byte	0A	3	Reg8 ← Reg8 OR Reg8
	0A	9+EA	Reg8 ← Reg8 OR Mem8
	08	16+EA	Mem8 ← Mem8 OR Reg8
Word	0B	3	Reg16 ← Reg16 OR Reg 16
	0B	9+EA	Reg16 ← Reg16 OR Mem16
	09	16+EA	Mem16 ← Mem16 OR Reg16

Immed to AX/AL



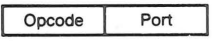
Opcode	Clocks	Operation
0C	4	AL ← AL OR Immed8
0D	4	AX ← AX OR Immed16

Immed to Memory/Reg

	Opcode	Clocks	Operation
Byte	80	4	Reg8 ← Reg8 OR Immed8
	80	17+EA	Mem8 ← Mem8 OR Immed8
Word	81	4	Reg16 ← Reg16 OR Immed16
	81	17+EA	Mem16 ← Mem16 OR Immed16

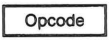
OUT = Output Byte, Word

Fixed port



	Opcode	Clocks	Operation
Byte	E6	10	Port8 ← AL
	E7	10	Port8 ← AX

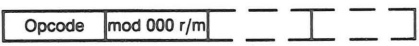
Variable port



	Opcode	Clocks	Operation
Word	EE	8	Port16 (in DX) ← AL
	EF	8	Port16 (in DX) ← AX

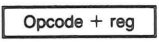
POP = Pop a Word from the Stack

Word Memory



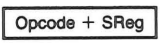
Opcode	Clocks	Operation
8F	17+EA	Mem16 ← (SP)++

Word Register



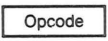
Opcode	Clocks	Operation
58+reg	8	Reg16 ← (SP)++

Segment Register



Opcode	Clocks	Operation
07+SReg	8	SReg ← (SP)++

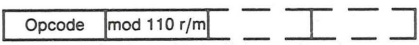
POPF = Pop the TOS into the Flags



Opcode	Clocks	Operation
9D	8	FLAGS ← (SP)++

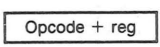
PUSH = Push a Word onto the Stack

Memory/Reg



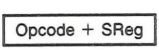
Opcode	Clocks	Operation
FF	16+EA	—(SP) ← Mem16

Word Register



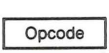
Opcode	Clocks	Operation
50+reg	11	—(SP) ← Reg16

Segment Register



Opcode	Clocks	Operation
06+SReg	10	—(SP) ← SReg

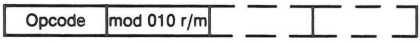
PUSHF = Push the Flags to the Stack



Opcode	Clocks	Operation
9C	10	—(SP) ← FLAGS

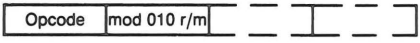
RCL = Rotate Left Through Carry

Memory or Reg by 1



	Opcode	Clocks	Operation
Byte	D0	2	rotate Reg 8 by 1
	D0	15+EA	rotate Mem8 by 1
Word	D1	2	rotate Reg 16 by 1
	D1	15+EA	rotate Mem16 by 1

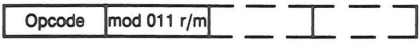
Memory or Reg by count in CL



	Opcode	Clocks	Operation
Byte	D2	8+4/bit	rotate Reg8 by CL
	D2	20+EA+4/bit	rotate Mem8 by CL
Word	D3	8+4/bit	rotate Reg16 by CL
	D3	20+EA+4/bit	rotate Mem16 by CL

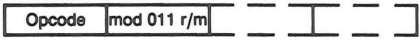
RCR = Rotate Right Through Carry

Memory or Reg by 1



	Opcode	Clocks	Operation
Byte	D0	2	rotate Reg8 by 1
	D0	15+EA	rotate Mem8 by 1
Word	D1	2	rotate Reg16 by 1
	D1	15+EA	rotate Mem16 by 1

Memory or Reg by count in CL



	Opcode	Clocks	Operation
Byte	D2	8+4/bit	rotate Reg8 by CL
	D2	20+EA+4/bit	rotate Mem8 by CL
Word	D3	8+4/bit	rotate Reg16 by CL
	D3	20+EA+4/bit	rotate Mem16 by CL

REP*x* = Repeat Prefix

Opcode

Opcode	Clocks	Operation	REP*x* =
F3	2	repeat next instruction until CX=0	REP
F3	2	repeat next instruction until CX=0 or ZF=1	REPE REPZ
F2	2	repeat next instruction until CX=0 or ZF=0	REPNE REPNZ

RET = Return from Subroutine

Opcode

Opcode	Clocks	Operation
C3	8	intra-segment return
CB	18	inter-segment return

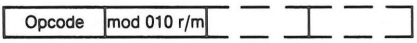
Return and add constant to SP

Opcode	DataL	DataH

Opcode	Clocks	Operation
C2	12	intra-segment ret and add
CA	17	inter-segment ret and add

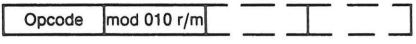
ROL = Rotate Left

Memory or Reg by 1



	Opcode	Clocks	Operation
Byte	D0	2	rotate Reg8 by 1
	D0	15+EA	rotate Mem8 by 1
Word	D1	2	rotate Reg16 by 1
	D1	15+EA	rotate Mem16 by 1

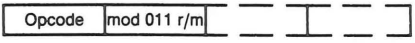
Memory or Reg by count in CL



	Opcode	Clocks	Operation
Byte	D2	8+4/bit	rotate Reg8 by CL
	D2	20+Ea+4/bit	rotate Mem8 by CL
Word	D3	8+4/bit	rotate Reg16 by CL
	D3	20+EA+4/bit	rotate Mem16 by CL

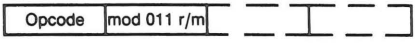
ROR = Rotate Right

Memory or Reg by 1



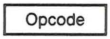
	Opcode	Clocks	Operation
Byte	D0	2	rotate Reg8 by 1
	D0	15+EA	rotate Mem8 by 1
Word	D1	2	rotate Reg16 by 1
	D1	15+EA	rotate Mem16 by 1

Memory or Reg by count in CL



	Opcode	Clocks	Operation
Byte	D2	8+4/bit	rotate Reg8 by CL
	D2	20+EA+4/bit	rotate Mem8 by CL
	D3	8+4/bit	rotate Reg16 by CL
	D3	20+EA+4/bit	rotate Mem16 by CL

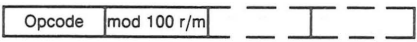
SAHF = Store AH in Flags



Opcode	Clocks	Operation
9E	4	copy AH to low byte of flags word

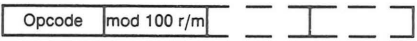
SAL/SHL = Arithmetic/Logical Left Shift

Memory or Reg by 1



	Opcode	Clocks	Operation
Byte	D0	2	shift Reg8 by 1
	D0	15+EA	shift Mem8 by 1
Word	D1	2	shift Reg16 by 1
	D1	15+EA	shift Mem16 by 1

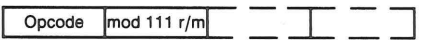
Memory or Reg by count in CL



	Opcode	Clocks	Operation
Byte	D2	8+4/bit	shift Reg8 by CL
	D2	20+EA+4/bit	shift Mem8 by CL
Word	D3	8+4/bit	shift Reg16 by CL
	D3	20+EA+4/bit	shift Mem16 by CL

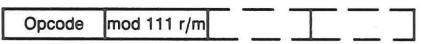
SAR = Arithmetic Right Shift

Memory or Reg by 1



	Opcode	Clocks	Operation
Byte	D0	2	shift Reg8 by 1
	D0	15+EA	shift Mem8 by 1
Word	D1	2	shift Reg16 by 1
	D1	15+EA	shift Mem16 by 1

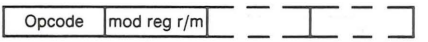
Memory or Reg by count in CL



	Opcode	Clocks	Operation
Byte	D2	8+4/bit	shift Reg8 by CL
	D2	20+EA+4/bit	shift Mem8 by CL
Word	D3	8+4/bit	shift Reg16 by CL
	D3	20+EA+4/bit	shift Mem16 by CL

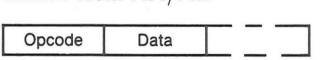
SBB = Integer Subtraction with Borrow

Memory/Reg with Reg



	Opcode	Clocks	Operation
Byte	1A	3	Reg8 ← Reg8 - Reg8 - CF
	1A	9+EA	Reg8 ← Reg8 - Mem8 - CF
	18	16+EA	Mem8 ← Mem8 - Reg8 - CF
Word	1B	3	Reg16 ← Reg16 - Reg16 - CF
	1B	9+EA	Reg16 ← Reg16 - Mem16 - CF
	19	16+EA	Mem16 ← Mem16 - Reg16 - CF

Immed from AX/AL



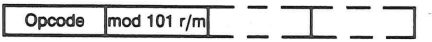
Opcode	Clocks	Operation
1C	4	AL ← AL - Immed8 - CF
1D	4	AX ← AX - Immed16 - CF

Immed from Memory/Reg

Opcode	Clocks	Operation
80	4	Reg8 ← Reg8 - Immed8 - CF
80	17+EA	Mem8 ← Mem8 - Immed8 - CF
81	4	Reg16 ← Reg16 - Immed16 - CF
81	17+EA	Mem16 ← Mem16 - Immed16 - CF
83	4	Reg16 ← Reg16 - Immed8 - CF
83	17+EA	Mem16 ← Mem16 - Immed8 - CF (Immed8 is sign-extended before subtract)

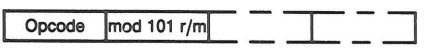
SHR = Logical Right Shift

Memory or Reg by 1



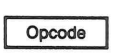
	Opcode	Clocks	Operation
Byte	D0	2	shift Reg8 by 1
	D0	15+EA	shift Mem8 by 1
Word	D1	2	shift Reg16 by 1
	D1	15+EA	shift Mem16 by 1

Memory or Reg by count in CL



	Opcode	Clocks	Operation
Byte	D2	8+4/bit	shift Reg8 by CL
	D2	20+Ea+4/bit	shift Mem8 by CL
Word	D3	8+4/bit	shift Reg16 by CL
	D3	20+EA+4/bit	shift Mem16 by CL

STC = Set Carry Flag



Opcode	Clocks	Operation
F9	2	set the carry flag

STD = Set Direction Flags

Opcode	Clocks	Operation
FD	2	set direction flag

STI = Set Interrupt Enable Flag

Opcode

Opcode	Clocks	Operation
FB	2	set interrupt flag

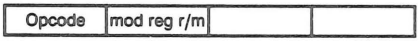
String = String Operations

Opcode

Opcode	Clocks	Operation	*String* =
A6	22	flags ← (SI) - (DI)	CMPS
A7	22	flags ← (SI) - (DI)	CMPS
A4	18	(DI) ← (SI)	MOVS
A5	18	(DI) ← (SI)	MOVS
AE	15	flags ← (DI) - AL	SCAS
AF	15	flags ← (DI) - AX	SCAS
AC	12	AL ← (SI)	LODS
AD	12	AX ← (SI)	LODS
AA	11	(DI) ← AL	STOS
AB	11	(DI) ← AX	STOS

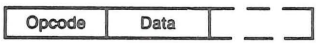
SUB = Integer Subtraction

Memory/Reg with Reg



	Opcode	Clocks	Operation
Byte	2A	3	Reg8 Reg8 - Reg8
	2A	9+EA	Reg8 Reg8 - Mem8
	28	16+EA	Mem8 Mem8 - Reg8
Word	2B	3	Reg16 Reg16 - Reg16
	2B	9+EA	Reg16 Reg16 - Mem16
	29	16+EA	Mem16 Mem16 - Reg16

Immed to AX/AL



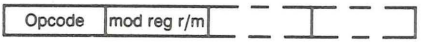
	Opcode	Clocks	Operation
Byte	2C	4	AL ← AL - Immed8
Word	2D	4	AX ← AX - Immed16

Immed to Memory/Reg

	Opcode	Clocks	Operation
Byte	80	4	Reg8 ← Reg8 - Immed8
	80	17+EA	Mem8 ← Mem8 - Immed8
Word	81	4	Reg16 ← Reg16 - Immed16
	81	17+EA	Mem16 ← Mem16 - Immed16
	83	4	Reg16 ← Reg16 - Immed8
	83	17+EA	Mem16 ← Mem16 - Immed8

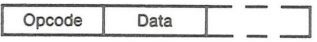
TEST = Logical Compare

Memory/Reg with Reg



	Opcode	Clocks	Operation
Byte	84	3	flags ← Reg8 AND Reg8
	84	9+EA	flags ← Reg8 AND Mem8
Word	85	3	flags ← Reg16 AND Reg16
	85	9+EA	flags ← Reg16 AND Mem16

Immed to AX/AL

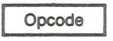


	Opcode	Clocks	Operation
Byte	A8	4	flags ← AL AND Immed8
Word	A9	4	flags ← AX AND Immed16

Immed to Memory/Reg

	Opcode	Clocks	Operation
Byte	F6	5	flags ← Reg8 AND Immed8
	F6	11+EA	flags ← Mem8 AND Immed8
Word	F7	5	flags ← Reg16 AND Immed16
	F7	11+EA	flags ← Mem16 AND Immed16

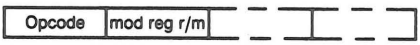
WAIT = Wait While TEST Pin Not Asserted



Opcode	Clocks	Operation
9B	3+5n	none

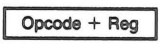
XCHG = Exchange Memory/Register with Register

Memory/Reg with Reg



	Opcode	Clocks	Operation
Byte	86	4	Reg8 ⇄ Reg8
	86	17+EA	Mem8 ⇄ Mem8
Word	87	4	Reg16 ⇄ Reg16
	87	17+EA	Mem16 ⇄ Mem16

Word Register with AX



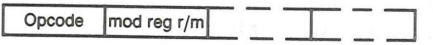
Opcode	Clocks	Operation
90+Reg	3	AX ⇄ Reg16

XLAT
XLATB = Table Look-up Translation

Opcode	Clocks	Operation
D7	11	replace AL with table entry

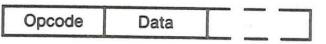
XOR = Logical Exclusive OR

Memory/Reg with Reg



	Opcode	Clocks	Operation
Byte	32	3	Reg8 ← Reg8 XOR Reg8
	32	9+EA	Reg8 ← Reg8 XOR Mem8
	30	16+EA	Mem8 ← Mem8 XOR Reg8
Word	33	3	Reg16 ← Reg16 XOR Reg16
	33	9+EA	Reg16 ← Reg16 XOR Mem16
	31	16+EA	Mem16 ← Mem16 XOR Reg16

Immed to AX/AL



Opcode	Clocks	Operation
34	4	AL ← AL XOR Immed8
35	4	AX ← AX XOR Immed16

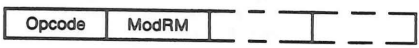
Immed to Memory/Reg

	Opcode	Clocks	Operation
Byte	80	4	Reg8 ← Reg8 XOR Immed8
	80	17+EA	Mem8 ← Mem8 XOR Immed8
Word	81	4	Reg16 ← Reg16 XOR Immed16
	81	17+EA	Mem16 ← Mem16 XOR Immed16

Notes for iAPX 186/188 Instructions

These instructions can be used only if the MOD186 control is specified. When MOD186 is specified, clocks for all instructions are as stated under "Clocks for MOD186 Operation."

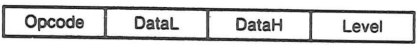
BOUND = Check Array Against Bounds



Opcode Operation

62 if Reg16<Mem16 at EA, or
Reg16>Mem16 at EA+2 then
INTERRUPT 5

ENTER = High Level Procedure Entry

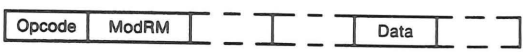


Opcode Operation

C8 build new stack frame

IMUL = Signed Multiplication

Mem/Reg* Immediate to Reg



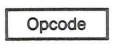
Opcode Operation

6B Reg 16 ← Reg 16 * Immed 8
6B Reg 16 ← Reg 16 * Immed 8
6B Reg 16 ← Mem 16 * Immed 8
69 Reg 16 ← Reg 16 * Immed 16
69 Reg 16 ← Reg 16 * Immed 16
69 Reg 16 ← Mem 16 * Immed 16

LEAVE = High Level Procedure Exit

Opcode	Operation
C9	release current stack frame and return to prior frame.

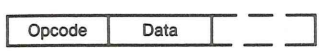
POPA = Pop All Registers



Opcode	Operation
61	restore registers from stack

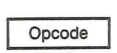
PUSH = Push a Word onto the Stack

Word Immediate



Opcode	Operation
6A	—(SP) ← Immed8 (sign extended)
68	—(SP) ← Immed16

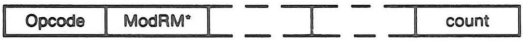
PUSHA = Push All Registers



Opcode	Operation
60	save registers on the stack

RCL = Rotate Left Through Carry

Mem or Reg by Immed8



*—(Reg field = 011)

Opcode	Operation
C0	rotate Reg8 by Immed8
C0	rotate Mem8 by Immed8
C1	rotate Reg16 by Immed8
C1	rotate Mem16 by Immed8

RCR = Rotate Right Through Carry

Mem or Reg by Immed8

*—(Reg field = 011)

Opcode	Operation
C0	rotate Reg8 by Immed8
C0	rotate Mem8 by Immed8
C1	rotate Reg16 by Immed8
C1	rotate Mem16 by Immed8

ROL = Rotate Left

Mem or Reg by Immed8

*—(Reg field = 000)

Opcode	Operation
C0	rotate Reg8 by Immed8
C0	rotate Mem8 by Immed8
C1	rotate Reg16 by Immed8
C1	rotate Mem16 by Immed8

ROR = Rotate Right

Mem or Reg by Immed8

*—(Reg field = 001)

Opcode	Operation
C0	rotate Reg8 by Immed8
C0	rotate Mem8 by Immed8
C1	rotate Reg16 by Immed8
C1	rotate Mem16 by Immed8

SAL/SHL = Arithmetic/Logical Left Shift

Mem or Reg by immediate count

*—(Reg field = 100)

Opcode	Operation
C0	rotate Reg8 by Immed8
C0	rotate Mem8 by Immed8
C1	rotate Reg16 by Immed8
C1	rotate Mem16 by Immed8

SAR = Arithmetic Right Shift

Mem or Reg by Immed8

*—(Reg field = 111)

Opcode	Operation
C0	rotate Reg8 by Immed8
C0	rotate Mem8 by Immed8
C1	rotate Reg16 by Immed8
C1	rotate Mem16 by Immed8

SHR = Logical Right Shift

Mem or Reg by Immed8

*—(Reg field = 101)

Opcode	Operation
C0	rotate Reg8 by Immed8
C0	rotate Mem8 by Immed8
C1	rotate Reg16 by Immed8
C1	rotate Mem16 by Immed8

String = String Operations (INS/OUTS)

Opcode	Clocks	Operation
6E	INS	(DI) ← port(DX)
6F	INS	(DI) ← port(DX:DX+1)
6C	OUTS	port(DX) ← (SI)
6D	OUTS	port(DX:DX+1) ← (SI)

Processor Reset Register Initialization

Flags = 0000H (to disable interrupts and single-stepping)

CS = FFFFH
IP = 0000H (to begin execution at FFFF0H)

DS = 0000H
SS = 0000H
ES = 0000H

No other registers are acted upon during reset.

MCS®-86 Reserved Locations

Reserved Memory Locations

Intel Corporation reserves the use of memory location FFFF0H through FFFFFH (with the exception of FFFF0H - FFFF5H for JMP instr.) for Intel hardware and software products. If you use these locations for some other purpose, you may preclude compatibility of your system with certain of these products.

Reserved Input/Output Locations

Intel Corporation reserves the use of input/output locations F8H through FFH for Intel hardware and software products. Users who wish to maintain compatibility with present and future Intel products should not use these locations.

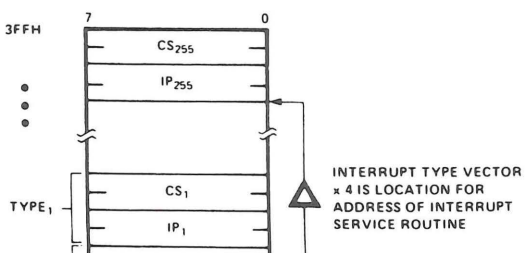
Reserved Interrupt Locations

Intel Corporation reserves the use of interrupts 0-31 (locations 00H through 7FH) for Intel hardware and software products. Users who wish to maintain compatibility with present and future Intel products should not use these locations.

Interrupts 0 through 4 (00H-13H) currently have dedicated hardware functions as defined below.

Interrupt	Location	Function
0	00H-03H	Divide by zero
1	04H-07H	Single step
2	08H-0BH	Non-maskable interrupt
3	0CH-0FH	One-byte interrupt instruction
4	10H-13H	Interrupt on overflow

Interrupt Pointer Table



iAPX 86/88/186/188 Instruction Set Matrix

Hi \ Lo	0	1	2	3	4	5	6	7
0	ADD b.f.r/m	ADD w.f.r/m	ADD b.t.r/m	ADD w.t.r/m	ADD b.ia	ADD w.ia	PUSH ES	POP ES
1	ADC b.f.r/m	ADC w.f.r/m	ADC b.t.r/m	ADC w.t.r/m	ADC b.i	ADC w.i	PUSH SS	POP SS
2	AND b.f.r/m	AND w.f.r/m	AND b.t.r/m	AND w.t.r/m	AND b.i	AND w.i	SEG ES	DAA
3	XOR b.f.r/m	XOR w.f.r/m	XOR b.t.r/m	XOR w.t.r/m	XOR b.i	XOR w.i	SEG SS	AAA
4	INC AX	INC CX	INC DX	INC BX	INC SP	INC BP	INC SI	INC DI
5	PUSH AX	PUSH CX	PUSH DX	PUSH BX	PUSH SP	PUSH BP	PUSH SI	PUSH DI
6	PUSHA	POPA	BOUND r,r/m					
7	JO	JNO	JB/ JNAE	JNB/ JAE	JE/ JZ	JNE/ JNZ	JBE/ JNA	JNBE/ JA
8	Immed b.r/m	Immed w.r/m	Immed b.r/m	Immed is.r/m	TEST b.r/m	TEST w.r/m	XCHG b.r/m	XCHG w.r/m
9	NOP	XCHG CX	XCHG DX	XCHG BX	XCHG SP	XCHG BP	XCHG SI	XCHG DI
A	MOV m → AL	MOV m → AX	MOV AL → m	MOV AX → m	MOVS b	MOVS w	CMPS b	CMPS w
B	MOV i → AL	MOV i → CL	MOV i → DL	MOV i → BL	MOV i → AH	MOV i → CH	MOV i → DH	MOV i → BH
C	Shift b,r/m,i	Shift w,r/m,i	RET (i-SP)	RET	LES	LDS	MOV b.i.r/m	MOV w.i.r/m
D	Shift b	Shift w	Shift b.v	Shift w.v	AAM	AAD		XLAT
E	LOOPNZ/ LOOPNE	LOOPZ/ LOOPE	LOOP	JCXZ	IN b	IN w	OUT b.	OUT w
F	LOCK		REP	REP Z	HLT	CMC	Grp1 b.r/m	Grp1 w.r/m

where

mod r/m	000	001	010	011	100	101	110	111
Immed	ADD	OR	ADC	SBB	AND	SUB	XOR	CMP
Shift	ROL	ROR	RCL	RCR	SHL/SAL	SHR	SHL/SAL	SAR
Grp1	TEST	—	NOT	NEG	MUL	IMUL	DIV	IDIV
Grp2	INC	DEC	CALL id	CALL I id	JMP id	JMP I id	PUSH	—

= 186 only instruction

iAPX 86/88/186/188 Instruction Set Matrix

Hi	Lo 8	9	A	B	C	D	E	F
0	OR b.f.r/m	OR w.f.r/m	OR b.t.r/m	OR w.t.r/m	OR b.i	OR w.i	PUSH CS	
1	SBB b.f.r/m	SBB w.f.r/m	SBB b.t.r/m	SBB w.t.r/m	SBB b.i	SBB w.i	PUSH DS	POP DS
2	SUB b.f.r/m	SUB w.f.r/m	SUB b.t.r/m	SUB w.t.r/m	SUB b.i	SUB w.i	SEG CS	DAS
3	CMP b.f.r/m	CMP w.f.r/m	CMP b.t.r/m	CMP w.t.r/m	CMP b.i	CMP w.i	SEG DS	AAS
4	DEC AX	DEC CX	DEC DX	DEC BX	DEC SP	DEC BP	DEC SI	DEC DI
5	POP AX	POP CX	POP DX	POP BX	POP SP	POP BP	POP SI	POP DI
6	PUSH iw	IMUL r,iw,r/m	PUSH is	IMUL r,is,r/m	INS b	INS w	OUTS b	OUTS w
7	JS	JNS	JP/ JPE	JNP/ JPO	JL/ JNGE	JNL/ JGE	JLE/ JNG	JNLE/ JG
8	MOV b.f.r/m	MOV w.f.r/m	MOV b.t.r/m	MOV w.t.r/m	MOV sr.f.r/m	LEA	MOV sr.t.r/m	POP r/m
9	CBW	CWD	CALL i.d	WAIT	PUSHF	POPF	SAHF	LAHF
A	TEST b,i	TEST w.i	STOS b	STOS w	LODS b	LODS w	SCAS b	SCAS w
B	MOV i → AX	MOV i → CX	MOV i → DX	MOV i → BX	MOV i → SP	MOV i → BP	MOV i → SI	MOV i → DI
C	ENTER iw,ib	LEAVE	RET I.(i-SP)	RET I	INT Type 3	INT (Any)	INTO	IRET
D	ESC 0	ESC 1	ESC 2	ESC 3	ESC 4	ESC 5	ESC 6	ESC 7
E	CALL d	JMP d	JMP i.d	JMP si.d	IN v.b	IN v.w	OUT v.d	OUT v.w
F	CLC	STC	CLI	STI	CLD	STD	Grp 2 b.r/m	Grp 2 w.r/m

b = byte operation
d = direct
f = from CPU reg
i = immediate
ia = immed. to accum.
ib = immediate byte
id = indirect
is = immed. byte sign ext.
iw = immediate word
l = long ie. intersegment
m = memory
r = register
r/m = EA is second byte
si = short intrasegment
sr = segment register
t = to CPU reg
v = variable
w = word operation
z = zero

Clocks for MOD186 Operation

FUNCTION	FORMAT	186 Clock Cycles
DATA TRANSFER		
MOV = Move:		
Register to Register Memory	1 0 0 0 1 0 0 w | mod reg r/m	2/12
Register memory to register	1 0 0 0 1 0 1 w | mod reg r/m	2/9
Immediate to register memory	1 1 0 0 0 1 1 w | mod 0 0 0 r/m | data | data if w =	12–13
Immediate to register	1 0 1 1 w reg | data | data if w	3–4
Memory to accumulator	1 0 1 0 0 0 0 w | addr-low | addr-high	9
Accumulator to memory	1 0 1 0 0 0 1 w | addr-low | addr-high	8
Register memory to segment register	1 0 0 0 1 1 1 0 | mod 0 reg r/m	2/9
Segment register to register memory	1 0 0 0 1 1 0 0 | mod 0 reg r/m	2/11
PUSH = Push:		
Memory	1 1 1 1 1 1 1 1 | mod 1 1 0 r/m	16
Register	0 1 0 1 0 reg	10
Segment register	0 0 0 reg 1 1 0	9
Immediate	0 1 1 0 1 0 s 0 | data | data if s = 0	10
PUSHA = Push All	0 1 1 0 0 0 0 0	36
POP = Pop:		
Memory	1 0 0 0 1 1 1 1 | mod 0 0 0 r/m	20
Register	0 1 0 1 1 reg	10
Segment register	0 0 0 reg 1 1 1 (reg ≠ 01)	8
POPA = Pop All	0 1 1 0 0 0 0 1	51
XCHG = Exchange:		
Register memory with register	1 0 0 0 0 1 1 w | mod reg r/m	4/17
Register with accumulator	1 0 0 1 0 reg	3
IN = Input from:		
Fixed port	1 1 1 0 0 1 0 w | port	10
Variable port	1 1 1 0 1 1 0 w	8
OUT = Output to:		
Fixed port	1 1 1 0 0 1 1 w | port	9
Variable port	1 1 1 0 1 1 1 w	7
XLAT – Translate byte to AL	1 1 0 1 0 1 1 1	11
LEA – Load EA to register	1 0 0 0 1 1 0 1 | mod reg r/m	6
LDS – Load pointer to DS	1 1 0 0 0 1 0 1 | mod reg r/m (mod ≠ 11)	18
LES – Load pointer to ES	1 1 0 0 0 1 0 0 | mod reg r/m (mod ≠ 11)	18
LAHF – Load AH with flags	1 0 0 1 1 1 1 1	2
SAHF – Store AH into flags	1 0 0 1 1 1 1 0	3
PUSHF – Push flags	1 0 0 1 1 1 0 0	9
POPF – Pop flags	1 0 0 1 1 1 0 1	8

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

Clocks for MOD186 Operation

FUNCTION	FORMAT	186 Clock Cycles
ARITHMETIC		
ADD = Add:		
Reg memory with register to either	0 0 0 0 0 0 d w | mod reg r/m	3/10
Immediate to register memory	1 0 0 0 0 0 s w | mod 0 0 0 r/m | data | data if s w = 01	4/16
Immediate to accumulator	0 0 0 0 0 1 0 w | data | data if w = 1	3/4
ADC = Add with carry:		
Reg memory with register to either	0 0 0 1 0 0 d w | mod reg r/m	3/10
Immediate to register memory	1 0 0 0 0 0 s w | mod 0 1 0 r/m | data | data if s w = 01	4/16
Immediate to accumulator	0 0 0 * 0 * 1 0 w | data | data if w = 1	3/4
INC = Increment:		
Register memory	* * * * * * * w | mod 0 0 0 r/m	3/15
Register	0 * 0 0 0 reg	3
SUB = Subtract:		
Reg memory and register to either	0 0 1 0 * 0 d w | mod reg r/m	3/10
Immediate from register memory	1 0 0 0 0 0 s w | mod * 0 1 r/m | data | data if s w = 01	4/16
Immediate from accumulator	0 0 1 0 * * 0 w | data | data if w = 1	3/4
SBB = Subtract with borrow:		
Reg memory and register to either	0 0 0 1 * 0 d w | mod reg r/m	3/10
Immediate from register memory	1 0 0 0 0 0 s w | mod 0 * * r/m | data | data if s w = 01	4/16
Immediate from accumulator	0 0 0 1 1 1 0 w | data | data if w = 1	3/4
DEC = Decrement:		
Register memory	1 1 1 1 * 1 1 w | mod 0 0 1 r/m	3/15
Register	0 1 0 0 1 reg	3
CMP = Compare:		
Register memory with register	0 0 1 1 1 0 1 w | mod reg r/m	3/10
Register with register memory	0 0 1 1 1 0 0 w | mod reg r/m	3/10
Immediate with register memory	1 0 0 0 0 0 s w | mod 1 1 1 r/m | data | data if s w = 01	3/10
Immediate with accumulator	0 0 1 1 1 1 0 w | data | data if w = 1	3/4
NEG = Change sign	1 1 1 1 0 1 1 w | mod 0 1 1 r/m	3
AAA – ASCII adjust for add	0 0 1 1 0 1 1 1	8
DAA – Decimal adjust for add	0 0 1 0 0 1 1 1	4
AAS – ASCII adjust for subtract	0 0 1 1 1 1 1 1	7
DAS – Decimal adjust for subtract	0 0 1 0 1 1 1 1	4
MUL – Multiply (unsigned)	1 1 1 1 0 1 1 w | mod 1 0 0 r/m	
Register-Byte		26–28
Register-Word		35–37
Memory-Byte		32–34
Memory-Word		41–43
IMUL – Integer multiply (signed)	1 1 1 1 0 1 1 w | mod 1 0 1 r/m	
Register-Byte		25–28
Register-Word		34–37
Memory-Byte		31–34
Memory-Word		40–43
IMUL – Integer immediate multiply (signed)	0 1 1 0 1 0 s 1 | mod reg r/m | data | data if s = 0	22–25/20–32
DIV – Divide (unsigned)	1 1 1 1 0 1 1 w | mod 1 1 0 r/m	
Register-Byte		29
Register-Word		38
Memory-Byte		35
Memory-Word		44

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

Clocks for MOD186 Operation

FUNCTION	FORMAT	186 Clock Cycles
ARITHMETIC (Continued):		
IDIV = Integer divide (signed)	1 1 1 1 0 1 1 w | mod 1 1 1 r/m	
Register-Byte		44–52
Register-Word		53–61
Memory-Byte		50–58
Memory-Word		59–67
AAM = ASCII adjust for multiply	1 1 0 1 0 1 0 0 | 0 0 0 0 1 0 1 0	19
AAD = ASCII adjust for divide	1 1 0 1 0 1 0 1 | 0 0 0 0 1 0 1 0	15
CBW = Convert byte to word	1 0 0 1 1 0 0 0	2
CWD = Convert word to double word	1 0 0 1 1 0 0 1	4
LOGIC		
Shift/Rotate Instructions:		
Register Memory by 1	1 1 0 1 0 0 0 w | mod TTT r/m	2/15
Register Memory by CL	1 1 0 1 0 0 1 w | mod TTT r/m	5 + n/17 + n
Register Memory by Count	1 1 0 0 0 0 0 w | mod TTT r/m | count	5+n/17+n
	TTT Instruction 0 0 0 ROL 0 0 1 ROR 0 1 0 RCL 0 1 1 RCR 1 0 0 SHL SAL 1 0 1 SHR 1 1 1 SAR	
AND = And:		
Reg memory and register to either	0 0 1 0 0 0 d w | mod reg r/m	3/10
Immediate to register memory	1 0 0 0 0 0 0 w | mod 1 0 0 r/m | data | data if w = 1	4/16
Immediate to accumulator	0 0 1 0 0 1 0 w | data | data if w = 1	3/4
TEST = And function to flags, no result:		
Register memory and register	1 0 0 0 0 1 0 w | mod reg r/m	3/10
Immediate data and register memory	1 1 1 1 0 1 1 w | mod 0 0 0 r/m | data | data if w = 1	4/10
Immediate data and accumulator	1 0 1 0 1 0 0 w | data | data if w = 1	3/4
OR = Or:		
Reg memory and register to either	0 0 0 0 1 0 d w | mod reg r/m	3/10
Immediate to register memory	1 0 0 0 0 0 0 w | mod 0 0 1 r/m | data | data if w = 1	4/16
Immediate to accumulator	0 0 0 0 1 1 0 w | data | data if w = 1	3/4
XOR = Exclusive or:		
Reg memory and register to either	0 0 1 1 0 0 d w | mod reg r/m	3/10
Immediate to register memory	1 0 0 0 0 0 0 w | mod 1 1 0 r/m | data | data if w = 1	4/16
Immediate to accumulator	0 0 1 1 0 1 0 w | data | data if w = 1	3/4
NOT = Invert register memory	1 1 1 1 0 1 1 w | mod 0 1 0 r/m	3
STRING MANIPULATION:		
MOVS = Move byte word	1 0 1 0 0 1 0 w	14
CMPS = Compare byte word	1 0 1 0 0 1 1 w	22
SCAS = Scan byte word	1 0 1 0 1 1 1 w	15
LODS = Load byte wd to AL AX	1 0 1 0 1 1 0 w	12
STOS = Stor byte wd from AL A	1 0 1 0 1 0 1 w	10
INS = Input byte/wd from DX port	0 1 1 0 1 1 0 w	14
OUTS = Output byte/wd to DX port	0 1 1 0 1 1 1 w	14

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

Clocks for MOD186 Operation

FUNCTION	FORMAT	186 Clock Cycles
STRING MANIPULATION (Continued):		
Repeated by count in CX		
MOVS Move string	1 1 1 1 0 0 1 0 | 1 0 1 0 0 1 0 w	8+8n
CMPS Compare string	1 1 1 1 0 0 1 z | 1 0 1 0 0 1 1 w	5+22n
SCAS Scan string	1 1 1 1 0 0 1 z | 1 0 1 0 1 1 1 w	5+15n
LODS Load string	1 1 1 1 0 0 1 0 | 1 0 1 0 1 1 0 w	6+11n
STOS Store string	1 1 1 1 0 0 1 0 | 1 0 1 0 1 0 1 w	6+9n
INS = Input string	1 1 1 1 0 0 1 0 | 0 1 1 0 1 1 0 w	8+8n
OUTS = Output string	1 1 1 1 0 0 1 0 | 0 1 1 0 1 1 1 w	8+8n
CONTROL TRANSFER		
CALL = Call:		
Direct within segment	* * * 0 * 0 0 0 | disp-low | disp-high	14
Register memory indirect within segment	* * * * * * * * | mod 0 1 0 r/m	13/19
Direct intersegment	* 0 0 * * 0 * 0 | segment offset | segment selector	23
Indirect intersegment	* * * * * * * * | mod 0 1 1 r/m (mod ≠ 11)	38
JMP = Unconditional jump:		
Short long	* * * 0 * 0 * * | disp-low	13
Direct within segment	* * * 0 * 0 0 * | disp-low | disp-high	13
Register memory indirect within segment	* * * * * * * * | mod 1 0 0 r/m	11/17
Direct intersegment	1 1 1 0 * 1 0 * | segment offset | segment selector	13
Indirect intersegment	1 1 1 1 1 1 1 1 | mod 1 0 1 r/m (mod ≠ 11)	26
RET = Return from CALL.		
Within segment	1 1 0 0 0 0 1 1	16
Within seg adding immed to SP	1 1 0 0 0 0 1 0 | data-low | data-high	18
Intersegment	1 1 0 0 1 0 1 1	22
Intersegment adding immediate to SP	1 1 0 0 1 0 1 0 | data-low | data-high	25

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

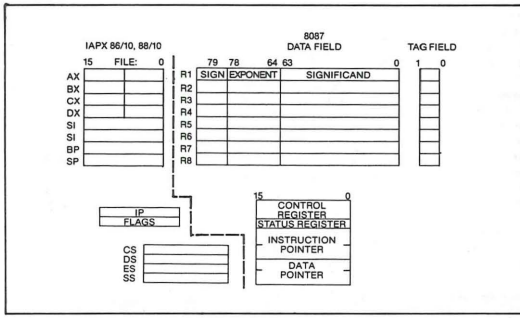
Clocks for MOD186 Operation

FUNCTION	FORMAT	186 Clock Cycles
CONTROL TRANSFER (Continued):		
JE/JZ = Jump on equal zero	0 1 1 1 0 1 0 0 | disp	4/13
JL/JNGE = Jump on less not greater or equal	0 1 1 1 1 1 0 0 | disp	4/13
JLE/JNG = Jump on less or equal not greater	0 1 1 1 1 1 1 0 | disp	4/13
JB/JNAE = Jump on below not above or equal	0 1 1 1 0 0 1 0 | disp	4/13
JBE/JNA = Jump on below or equal not above	0 1 1 1 0 1 1 0 | disp	4/13
JP/JPE = Jump on parity parity even	0 1 1 1 1 0 1 0 | disp	4/13
JO = Jump on overflow	0 1 1 1 0 0 0 0 | disp	4/13
JS = Jump on sign	0 1 1 1 1 0 0 0 | disp	4/13
JNE/JNZ = Jump on not equal not zero	0 1 1 1 0 1 0 1 | disp	4/13
JNL/JGE = Jump on not less greater or equal	0 1 1 1 1 1 0 1 | disp	4/13
JNLE/JG = Jump on not less or equal greater	0 1 1 1 1 1 1 1 | disp	4/13
JNB/JAE = Jump on not below above or equal	0 1 1 1 0 0 1 1 | disp	4/13
JNBE/JA = Jump on not below or equal above	0 1 1 1 0 1 1 1 | disp	4/13
JNP/JPO = Jump on not par par odd	0 1 1 1 1 0 1 1 | disp	4/13
JNO = Jump on not overflow	0 1 1 1 0 0 0 1 | disp	4/13
JNS = Jump on not sign	0 1 1 1 1 0 0 1 | disp	4/13
LOOP = Loop CX times	1 1 1 0 0 0 1 0 | disp	5/15
LOOPZ/LOOPE = Loop while zero equal	1 1 1 0 0 0 0 1 | disp	6/16
LOOPNZ/LOOPNE = Loop while not zero equal	1 1 1 0 0 0 0 0 | disp	6/16
JCXZ = Jump on CX zero	1 1 1 0 0 0 1 1 | disp	16 5
ENTER = Enter Procedure	1 1 0 0 1 0 0 0 | data-low | data-high | L	
L = 0		15
L = 1		25
L > 1		22+16(n−1)
LEAVE = Leave Procedure	1 1 0 0 1 0 0 1	8
INT = Interrupt:		
Type specified	1 1 0 0 1 1 0 1 | type	47
Type 3	1 1 0 0 1 1 0 0	45
INTO = Interrupt on overflow	1 1 0 0 1 1 1 0	48/4
IRET = Interrupt return	1 1 0 0 1 1 1 1	28
BOUND = Detect value out of range	0 1 1 0 0 0 1 0 | mod reg r m	33–35

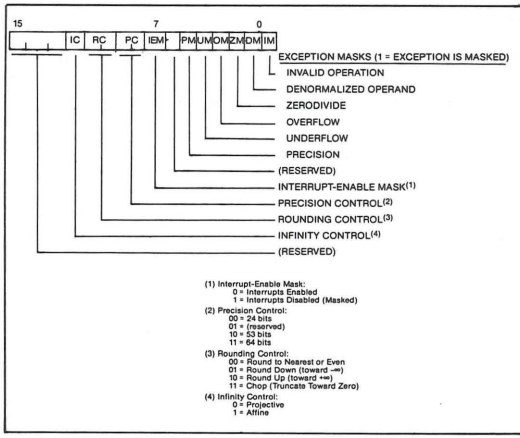
Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

Appendix
8087 Expanded Register Model

Expanded Register Set



Control Word Format

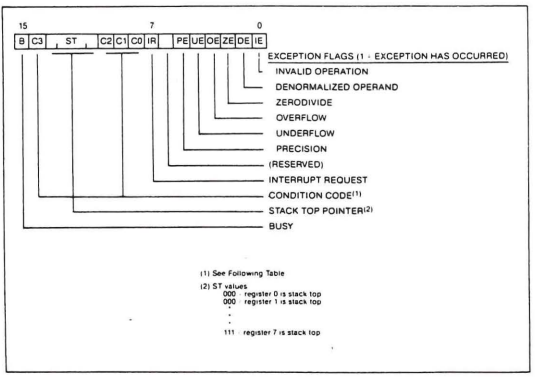


Exception Response Summary

Exception	Masked Response	Unmasked Response
Invalid Operation	If one operand is NAN; return it; if both are NANs, return, NAN with larger absolute value; if neither is NAN, return *indefinite*.	Request interrupt
Zerodivide	Return ∞ signed with "exclusive or" of operand signs	Request interrupt
Denormalized	Memory operand; preceed as usual. Register operand: convert to valid unnormal, then re-evaluate for exceptions.	Request interrupt.
Overflow	Return properly signed ∞.	Register destination, adjust exponent*, store result, request interrupt. Memory destination: request interrupt
Underflow	Denormalize result.	Register destination; adjust exponent*, store result, request interrupt. Memory destination request interrupt
Precision	Return rounded result.	Return rounded result, request interrupt.

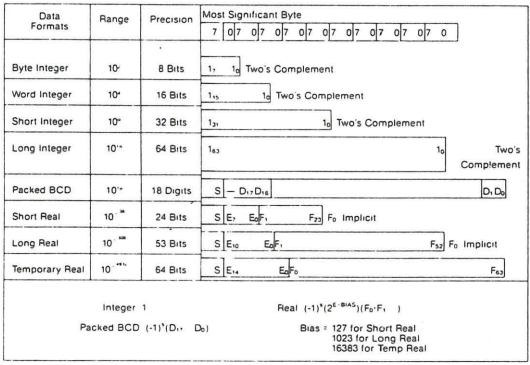
* On overflow, 24,576 decimal is *subtracted* from the true result's exponent; this forces the exponent back into range and permits a user exception handler to ascertain the true result from the adjusted result that is returned. On underflow, the same constant is *added* to the true result's exponent.

STATUS WORD FORMAT



8087 Data Types and Storage Formats

8087 Data Types



8087 Storage Allocation Directives

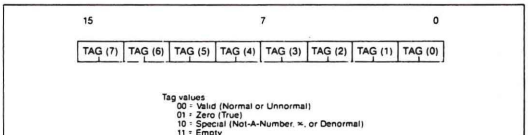
Directive	Interpretation	8087 Data Types
DW	Define Word	Word integer
DD	Define Doubleword	Short integer, short real
DQ	Define Quadword	Long integer, long real
DT	Define Tenbyte	Packed decimal, temporary real

Condition Code Interpretation

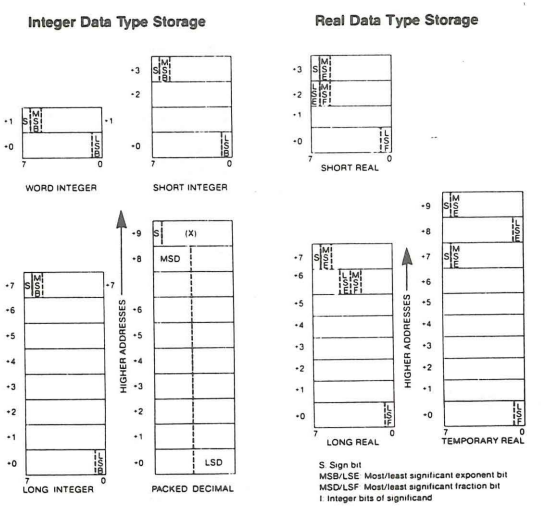
Instruction	C_3	C_2	C_1	C_0	Interpretation
Compare, Test	0	X	X	0	A > B
	0	X	X	1	A < B
	1	X	X	0	A = B
	1	X	X	1	A ? B (not comparable)
Remainder	Q_1	0	Q_0	Q_2	Complete reduction
	Q_1	1	Q_0	Q_2	Incomplete reduction
Examine	0	0	0	0	Valid, positive, unnormalized
	0	0	0	1	Invalid, positive, exponent ≠ 0
	0	0	1	0	Valid, negative, unnormalized
	0	0	1	1	Invalid, negative, exponent ≠ 0
	0	1	0	0	Valid, positive, normalized
	0	1	0	1	Infinity, positive
	0	1	1	1	Valid, negative, normalized
	0	1	0	1	Infinity, negative
	1	0	0	0	Zero, positive
	1	0	0	1	Empty
	1	0	1	0	Zero, negative
	1	0	1	1	Empty
	1	1	0	0	Invalid, positive, exponent ≠ 0
	1	1	0	1	Empty
	1	1	1	0	Invalid, negative, exponent ≠ 0
	1	1	1	1	Empty

X = value is not affected by instruction
Q = C_0 C_3 C_1, hold 3 LSBs of the quotient generated during a remainder operation

Tag Word Format



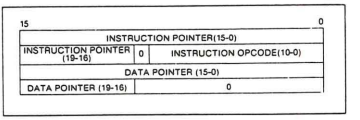
Integer Data Type Storage
Real Data Type Storage



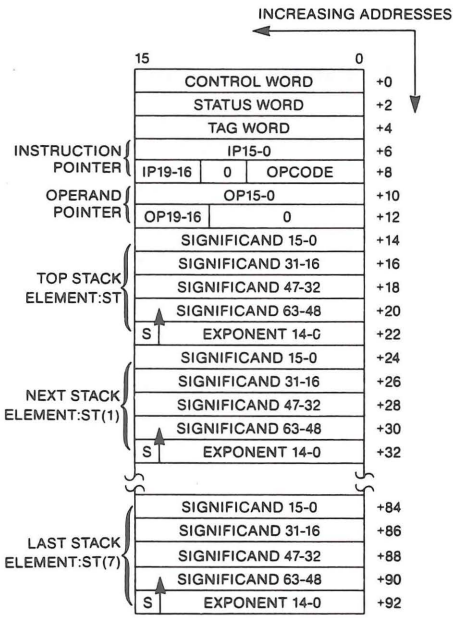
S: Sign bit
MSB/LSB: Most/least significant bit
MSD/LSD: Most/least significant decimal digit
(X): Bits have no significance

S: Sign bit
MSB/LSE: Most/least significant exponent bit
MSD/LSF: Most/least significant fraction bit
I: Integer bits of significand

Instruction and Data Pointer Storage

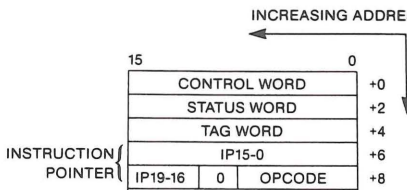


FSAVE/FRSTOR Storage



NOTES:
S = Sign
Bit 0 of each field is rightmost, least significant bit of corresponding register field.
Bit 63 of significand is integer bit (assumed binary point is immediately to the right).

FSTENV/FLDENV Storage



8087 Instruction Encoding and Decoding

8087 Instruction Encoding Guide

	Lower-addressed Byte								Higher-addressed Byte								0, 1, or 2 bytes
[1]	1	1	0	1	1	OP-A			1	MOD			1	OP-B		R/M	DISPLACEMENT
[2]	1	1	0	1	1	FORMAT			OP-A	MOD			OP-B			R/M	DISPLACEMENT
[3]	1	1	0	1	1	R	P		OP-A	1	1		OP-B			REG	
[4]	1	1	0	1	1	0	0	1	1	1	1	OP					
[5]	1	1	0	1	1	0	0	1	1	1	1	OP					
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	

[1] Memory transfers, including applicable processor control instructions; 0, 1, or 2 displacement bytes may follow
[2] Memory arithmetic and comparison instructions; 0, 1, or 2 displacement bytes may follow
[3] Stack arithmetic and comparison instructions
[4] Constant, transcendental, some arithmetic instructions
[5] Processor control instructions that do not reference memory

OP, OP-A, OP-B: Instruction opcode, possibly split into two fields
MOD: Same as CPU mode field; see table 4-8
R/M: Same as CPU register/memory field; see table 4-10

FORMAT: Defines memory operand
- 00 = short real
- 01 = short integer
- 10 = long real
- 11 = word integer

R: 0 = return result to stack top
- 1 = return result to other register

P: 0 = do not pop stack
- 1 = pop stack after operation

REG: register stack element
- 000 = stack top
- 001 = next on stack
- 010 = third stack element, etc

8087 Instruction Decoding Guide

1st Byte			2nd Byte		Bytes 3, 4	ASM-86 Instruction Format	
Hex	Binary						
D8	1101	1000	MOD00	0R/M	(disp-lo),(disp-hi)	FADD	short-real
D8	1101	1000	MOD00	1R/M	(disp-lo),(disp-hi)	FMUL	short-real
D8	1101	1000	MOD01	0R/M	(disp-lo),(disp-hi)	FCOM	short-real
D8	1101	1000	MOD01	1R/M	(disp-lo),(disp-hi)	FCOMP	short-real
D8	1101	1000	MOD10	0R/M	(disp-lo),(disp-hi)	FSUB	short-real
D8	1101	1000	MOD10	1R/M	(disp-lo),(disp-hi)	FSUBR	short-real
D8	1101	1000	MOD11	0R/M	(disp-lo),(disp-hi)	FDIV	short-real
D8	1101	1000	MOD11	1R/M	(disp-lo),(disp-hi)	FDIVR	short-real
D8	1101	1000	1100	0REG		FADD	ST,ST(i)
D8	1101	1000	1100	1REG		FMUL	ST,ST(i)
D8	1101	1000	1101	0REG		FCOM	ST(i)
D8	1101	1000	1101	1REG		FCOMP	ST(i)
D8	1101	1000	1110	0REG		FSUB	ST,ST(i)
D8	1101	1000	1110	1REG		FSUBR	ST,ST(i)
D8	1101	1000	1111	0REG		FDIV	ST,ST(i)
D8	1101	1000	1111	1REG		FDIVR	ST,ST(i)
D9	1101	1001	MOD00	0R/M	(disp-lo),(disp-hi)	FLD	short-real
D9	1101	1001	MOD00	1R/M		reserved	
D9	1101	1001	MOD01	0R/M	(disp-lo),(disp-hi)	FST	short-real
DA	1101	1010	MOD11	0R/M	(disp-lo),(disp-hi)	FIDIV	short-integer
DA	1101	1010	MOD11	1R/M	(disp-lo),(disp-hi)	FIDIVR	short-integer
DA	1101	1010	11--	----		reserved	
DB	1101	1011	MOD00	0R/M	(disp-lo),(disp-hi)	FILD	short-integer
DB	1101	1011	MOD00	1R/M	(disp-lo),(disp-hi)	reserved	
DB	1101	1011	MOD01	0R/M	(disp-lo),(disp-hi)	FIST	short-integer
DB	1101	1011	MOD01	1R/M	(disp-lo),(disp-hi)	FISTP	short-integer
DB	1101	1011	MOD10	0R/M	(disp-lo),(disp-hi)	reserved	
DB	1101	1011	MOD10	1R/M	(disp-lo),(disp-hi)	FLD	temp-real
DB	1101	1011	MOD11	0R/M	(disp-lo),(disp-hi)	reserved	
DB	1101	1011	MOD11	1R/M	(disp-lo),(disp-hi)	FSTP	temp-real
DB	1101	1011	110-	----		reserved	
DB	1101	1011	1110	0000		FENI	
DB	1101	1011	1110	0001		FDISI	
DB	1101	1011	1110	0010		FCLEX	
DB	1101	1011	1110	0011		FINIT	
DB	1101	1011	1110	01--		reserved	
DB	1101	1011	1110	1---		reserved	
BB	1101	1011	1111	----		reserved	
DC	1101	1100	MOD00	0R/M	(disp-lo),(disp-hi)	FADD	long-real
DC	1101	1100	MOD00	1R/M	(disp-lo),(disp-hi)	FMUL	long-real
DC	1101	1100	MOD01	0R/M	(disp-lo),(disp-hi)	FCOM	long-real
DC	1101	1100	MOD01	1R/M	(disp-lo),(disp-hi)	FCOMP	long-real
DC	1101	1100	MOD10	0R/M	(disp-lo),(disp-hi)	FSUBR	long-real
DC	1101	1100	MOD10	1R/M	(disp-lo),(disp-hi)	FSUB	long-real
DC	1101	1100	MOD11	0R/M	(disp-lo),(disp-hi)	FDIVR	long-real
DC	1101	1100	MOD11	1R/M	(disp-lo),(disp-hi)	FDIV	long-real
DC	1101	1100	1100	0REG		FADD	ST(i),ST
DC	1101	1100	1100	1REG		FMUL	ST(i),ST
DC	1101	1100	1101	0REG		*(2)	
DC	1101	1100	1101	1REG		*(3)	
DC	1101	1100	1110	0REG		FSUBR	ST(i),ST
DC	1101	1100	1110	1REG		FSUB	ST(i),ST
DC	1101	1100	1111	0REG		FDIVR	ST(i),ST
DC	1101	1100	1111	1REG		FDIV	ST(i),ST
DD	1101	1101	MOD00	0R/M	(disp-lo),(disp-hi)	FLD	long-real
DD	1101	1101	MOD00	1R/M		reserved	
DD	1101	1101	MOD01	0R/M	(disp-lo),(disp-hi)	FST	long-real
DD	1101	1101	MOD00	1R/M	(disp-lo),(disp-hi)	FSTP	long-real
DD	1101	1101	MOD10	0R/M	(disp-lo),(disp-hi)	FRSTOR	94-bytes
DD	1101	1101	MOD10	1R/M	(disp-lo),(disp-hi)	reserved	
DD	1101	1101	MOD11	0R/M	(disp-lo),(disp-hi)	FSAVE	94-bytes
DD	1101	1101	MOD11	1R/M	(disp-lo),(disp-hi)	FSTSW	2-bytes
DD	1101	1101	1100	0REG		FFREE	ST(i)
DD	1101	1101	1100	1REG		*(4)	
DD	1101	1101	1101	0REG		FST	ST(i)
DD	1101	1101	1101	1REG		FSTP	ST(i)

8087 Instruction Decoding Guide (cont.)

1st Byte			2nd Byte		Bytes 3, 4	ASM-86 Instruction Format	
Hex	Binary						
D9	1101	1001	MOD01	1R/M	(disp-lo),(disp-hi)	FSTP	short-real
D9	1101	1001	MOD10	0R/M	(disp-lo),(disp-hi)	FLDENV	14-bytes
D9	1101	1001	MOD10	1R/M	(disp-lo),(disp-hi)	FLDCW	2-bytes
D9	1101	1001	MOD11	0R/M	(disp-lo),(disp-hi)	FSTENV	14-bytes
D9	1101	1001	MOD11	1R/M	(disp-lo),(disp-hi)	FSTCW	2-bytes
D9	1101	1001	1100	0REG		FLD	ST(i)
D9	1101	1001	1100	1REG		FXCH	ST(i)
D9	1101	1001	1101	0000		FNOP	
D9	1101	1001	1101	0001		reserved	
D9	1101	1001	1101	001-		reserved	
D9	1101	1001	1101	01--		reserved	
D9	1101	1001	1101	1REG		*(1)	
D9	1101	1001	1110	0000		FCHS	
D9	1101	1001	1110	0001		FABS	
D9	1101	1001	1110	001-		reserved	
D9	1101	1001	1110	0100		FTST	
D9	1101	1001	1110	0101		FXAM	
D9	1101	1001	1110	011-		reserved	
D9	1101	1001	1110	1000		FLD1	
D9	1101	1001	1110	1001		FLDL2T	
D9	1101	1001	1110	1010		FLDL2E	
D9	1101	1001	1110	1011		FLDPI	
D9	1101	1001	1110	1100		FLDLG2	
D9	1101	1001	1110	1101		FLDLN2	
D9	1101	1001	1110	1110		FLDZ	
D9	1101	1001	1110	1111		reserved	
D9	1101	1001	1111	0000		F2XM1	
D9	1101	1001	1111	0001		FYL2X	
D9	1101	1001	1111	0010		FPTAN	
D9	1101	1001	1111	0011		FPATAN	
D9	1101	1001	1111	0100		FXTRACT	
D9	1101	1001	1111	0101		reserved	
D9	1101	1001	1111	0110		FDECSTP	
D9	1101	1001	1111	0111		FINCSTP	
D9	1101	1001	1111	1000		FPREM	
D9	1101	1001	1111	1001		FYL2XP1	
D9	1101	1001	1111	1010		FSQRT	
D9	1101	1001	1111	1011		reserved	
D9	1101	1001	1111	1100		FRNDINT	
D9	1101	1001	1111	1101		FSCALE	
D9	1101	1001	1111	111-		reserved	
DA	1101	1010	MOD00	0R/M	(disp-lo),(disp-hi)	FIADD	short-integer
DA	1101	1010	MOD00	1R/M	(disp-lo),(disp-hi)	FIMUL	short-integer
DA	1101	1010	MOD01	0R/M	(disp-lo),(disp-hi)	FICOM	short-integer
DA	1101	1010	MOD01	1R/M	(disp-lo),(disp-hi)	FICOMP	short-integer
DA	1101	1010	MOD10	0R/M	(disp-lo),(disp-hi)	FISUB	short-integer
DA	1101	1010	MOD10	1R/M	(disp-lo),(disp-hi)	FISUBR	short-integer

8087 Instruction Decoding Guide (cont.)

1st Byte		2nd Byte		Bytes 3, 4	ASM-86 Instruction Format	
Hex	Binary					
DD	1101 1101	111-	----		reserved	
DE	1101 1110	MOD00	0R/M	(disp-lo),(disp-hi)	FIADD	word-integer
DE	1101 1110	MOD00	1R/M	(disp-lo),(disp-hi)	FIMUL	word-integer
DE	1101 1110	MOD01	0R/M	(disp-lo),(disp-hi)	FICOM	word-integer
DE	1101 1110	MOD01	1R/M	(disp-lo),(disp-hi)	FICOMP	word-integer
DE	1101 1110	MOD10	0R/M	(disp-lo),(disp-hi)	FISUBR	word-integer
DE	1101 1110	MOD10	1R/M	(disp-lo),(disp-hi)	FISUB	word-integer
DE	1101 1110	MOD11	0R/M	(disp-lo),(disp-hi)	FIDIVR	word-integer
DE	1101 1110	MOD11	1R/M	(disp-lo),(disp-hi)	FIDIV	word-integer
DE	1101 1110	1100	0REG		FADDP	ST(i),ST
DE	1101 1110	1100	1REG		FMULP	ST(i),ST
DE	1101 1110	1101	0---		*(5)	
DE	1101 1110	1101	1000		reserved	
DE	1101 1110	1101	1001		FCOMPP	
DE	1101 1110	1101	101-		reserved	
DE	1101 1110	1101	11--		reserved	
DE	1101 1110	1110	0REG		FSUBRP	ST(i),ST
DE	1101 1110	1110	1REG		FSUBP	ST(i),ST
DE	1101 1110	1111	0REG		FDIVRP	ST(i),ST
DE	1101 1110	1111	1REG		FDIVP	ST(i),ST
DF	1101 1111	MOD00	0R/M	(disp-lo),(disp-hi)	FILD	word-integer
DF	1101 1111	MOD00	1R/M	(disp-lo),(disp-hi)	reserved	
DF	1101 1111	MOD01	0R/M	(disp-lo),(disp-hi)	FIST	word-integer
DF	1101 1111	MOD01	1R/M	(disp-lo),(disp-hi)	FISTP	word-integer
DF	1101 1111	MOD10	0R/M	(disp-lo),(disp-hi)	FBLD	packed-decimal
DF	1101 1111	MOD10	1R/M	(disp-lo),(disp-hi)	FILD	long-integer
DF	1101 1111	MOD11	0R/M	(disp-lo),(disp-hi)	FBSTP	packed-decimal
DF	1101 1111	MOD11	1R/M	(disp-lo),(disp-hi)	FISTP	long-integer
DF	1101 1111	1100	0REG		*(6)	
DF	1101 1111	1100	1REG		*(7)	
DF	1101 1111	1101	0REG		*(8)	
DF	1101 1111	1101	1REG		*(9)	
DF	1101 1111	111-	----		reserved	

*The marked encodings are *not* generated by the language translators. If, however, the 8087 encounters one one these encodings in the instruction stream, it will execute it as follows:

(1) FSTP ST(i)

(2) FCOM ST(i)

(3) FCOMP ST(i)

(4) FXCH ST(i)

(5) FCOMP ST(i)

(6) FFREE ST(i) and pop stack

(7) FXCH ST(i)

(8) FSTP ST(i)

(9) FSTP ST(i)

8087 Instruction Set

Notes for 8087 Instructions

The individual instruction descriptions are shown by a format box such as the following:

WAIT	op1	m/op/r/m	addr1	addr2

These are the byte-wise representations of the object code generated by the assembler and are interpreted as follows:

- WAIT is an 8086 wait instruction, NOP or emulator instruction.
- op1 is the opcode, possibly taking two bytes.
- m/op/r/m byte (middle 3-bits is part of the opcode).
- addr1 and addr2 are offsets of either 8 or 16 bits.

For integer functions, m = 0 for short-integer memory operand; 1 for word-integer memory operand.
For real functions, m = 0 for short-real memory operand; 1 for longreal memory operand.
i = stack element index.

If mod = 00 then DISP = 0 , disp-lo and disp-hi are absent.
If mod = 01 then DISP = disp-lo sign-extended to 16 bits, disp-hi is absent.
If mod = 10 then DISP = disp-hi; disp-lo.
If mod = 11 then r/m is treated as an ST(i) field.

If r/m = 000 then EA = (BX)+(SI)+DISP
If r/m = 001 then EA = (BX)+(DI)+DISP
If r/m = 010 then EA = (BP)+(SI)+DISP
If r/m = 011 then EA = (BP)+(DI)+DISP
If r/m = 100 then EA = (SI)+DISP
If r/m = 101 then EA = (DI)+DISP
If r/m = 110 then EA (BP)+DISP*
If r/m = 111 then EA = (BX)+DISP
*Except if mod = 000 and r/m = 110 then EA = disp-hi; disp-lo.

ST(0) = Current stack top
ST(i) = i[th] register below stack top
d = Destination
0 — Destination is ST(0)
1 — Destination is ST(i)
P = Pop
0 — No pop
1 — Pop ST(0)
R = Reverse
0 — Destination (op) source
1 — Source (op) destination

For FSQRT: $-0 \leq ST(0) \leq +\infty$
For FSCALE: $-2^{15} \leq ST(1) < +2^{15}$ and ST(1) integer
For F2XM1: $0 \leq ST(0) \leq 2^{-1}$
For FYL2X: $0 \leq ST(0) < \infty$
$-\infty < ST(1) < +\infty$
For FYL2XP1: $0 < |ST(0)| < (2-\sqrt{2})/2$
$-\infty \leq ST(1) < \infty$
For FPTAN: $0 \leq ST(0) < \pi/4$
For FPATAN: $0 \leq ST(0) < ST(1) < +\infty$

F2XMI = Compute $2^x - 1$

WAIT	op1	op2

8087 Encoding	Execution Clocks Typical Range	Operation
9B D9 F0	500 310-630	$ST \leftarrow 2^{ST}-1$

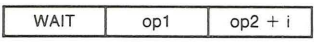
FABS = Absolute Value

WAIT	op1	op2

8087 Encoding	Execution Clocks Typical Range	Operation
9B D9 E1	14 10-17	$ST \leftarrow \|ST\|$

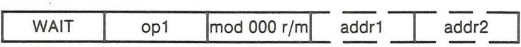
FADD = Add Real

Stack top + Stack element



8087 Encoding	Execution Clocks Typical Range	Operation
9B D8 C0+i	85 70-100	ST ← ST + ST(i)
9B DC C0+i	85 70-100	ST(i) ← ST + ST(i)

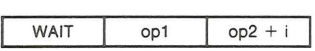
Stack top + memory operand



8087 Encoding	Execution Clocks Typical Range	Operation
9B D8 m0rm	105+EA (90-120)+EA	ST ← ST + mem-op (short-real)
9B DC m0rm	110+EA (95-125)+EA	ST ← ST + mem-op (long-real)

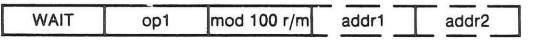
FADDP = Add Real and Pop

Stack top + Stack Element



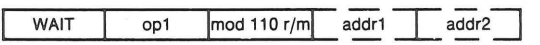
8087 Encoding	Execution Clocks Typical Range	Operation
9B DE C1	90 75-105	ST(1) ← ST + ST(1) pop stack
9B DE C0+i	90 75-105	ST(i) ← ST + ST(i) pop stack

FBLD = Packed Decimal (BCD) Load



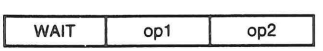
8087 Encoding	Execution Clocks Typical Range	Operation
9B DF m4rm	300+EA (290-310)+EA	push stack ST ← mem-op

FBSTP = Packed Decimal (BCD) Store and Pop



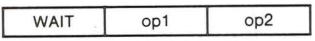
8087 Encoding	Execution Clocks Typical Range	Operation
9B DF m6rm	530+EA (520-540)+EA	mem-op ← ST pop stack

FCHS = Change Sign



8087 Encoding	Execution Clocks Typical Range	Operation
9B D9 E0	15 10-17	ST ← —ST

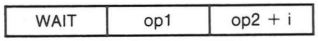
FCLEX
FNCLEX = Clear Exceptions



8087 Encoding	Execution Clocks Typical Range	Operation
9B DB E2	5 2-8	clear 8087 exceptions
90 DB E2	5 2-8	clear 8087 exceptions (no wait)

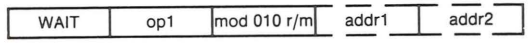
FCOM = Compare Real

Compare Stack top and Stack element



8087 Encoding	Execution Clocks Typical Range	Operation
9B D8 D1	45 40-50	ST — ST(1)
9B D8 D0+i	45 40-50	ST — ST(i)

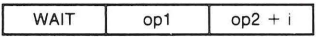
Compare Stack top and memory operands



8087 Encoding	Execution Clocks Typical Range	Operation
9B D8 m2rm	65+EA (60-70)+EA	ST — memop (short-real)
9B DC m2rm	70+EA (65-75)+EA	ST — memop (long-real)

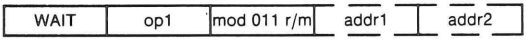
FCOMP = Compare Real and Pop

Compare Stack top and Stack element and pop



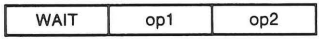
8087 Encoding	Execution Clocks Typical Range	Operation
9B D8 D9	47 42-52	ST — ST(1) pop stack
9B D8 D8+i	47 42-52	ST — ST(i) pop stack

Compare Stack top and memory operand and pop



8087 Encoding	Execution Clocks Typical Range	Operation
9B D8 m3rm	68+EA (63-73)+EA	ST — mem-op pop stack (short-real)
9B DC m3rm	72+EA (67-77)+EA	ST — mem-op pop stack (long-real)

FCOMPP = Compare Real and Pop Twice



8087 Encoding	Execution Clocks Typical Range	Operation
9B DE D9	50 45-55	ST — ST(1) pop stack pop stack

FDECSTP = Decrement Stack Pointer

WAIT	op1	op2

8087 Encoding	Execution Clocks Typical Range	Operation
9B D9 F6	9 6-12	stack pointer ← stack pointer 1

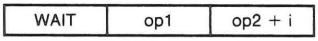
FDISI
FNDISI = Disable Interrupts

WAIT	op1	op2

8087 Encoding	Execution Clocks Typical Range	Operation
9B DB E1	5 2-8	Set 8087 interrupt mask
90 DB E1	5 2-8	Set 8087 interrupt mask (no wait)

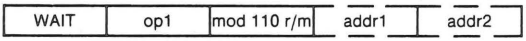
FDIV = Divide Real

Stack top and Stack element



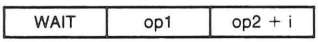
8087 Encoding	Execution Clocks Typical Range	Operation
9B D8 F0+i	198 193-203	ST ← ST/ST(i)
9B DC F8+i	198 193-203	ST(i) ← ST(i)/ST

Stack top and memory operand



8087 Encoding	Execution Clocks Typical Range	Operation
9B D8 m6rm	220+EA (215-225)+EA	ST ← ST/mem-op (short-real)
9B DC m6rm	225+EA (220-230)+EA	ST ← ST/mem-op (long-real)

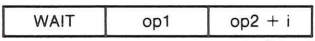
FDIVP = Divide Real and Pop



8087 Encoding	Execution Clocks Typical Range	Operation
9B DE F9	202 197-207	ST(1) ← ST(1)/ST pop stack
9B DE F8+i	202 197-207	ST(i) ← ST(i)/ST pop stack

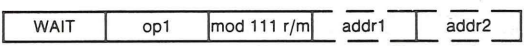
FDIVR = Divide Real Reversed

Stack top and Stack element



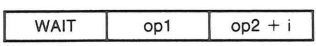
8087 Encoding	Execution Clocks Typical Range	Operation
9B D8 F8+i	199 194-204	ST ← ST(i)/ST
9B DC F0+i	199 194-204	ST(i) ← ST/ST(i)

Stack top and memory operand



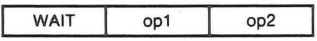
8087 Encoding	Execution Clocks Typical Range	Operation
9B D8 m7rm	221+EA (216-226)+EA	ST ← mem-op/ST (short-real)
9B DC m7rm	226+EA (221-231)+EA	ST ← mem-op/ST (long-real)

FDIVRP = Divide Real Reversed and Pop



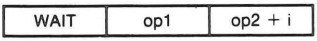
8087 Encoding	Execution Clocks Typical Range	Operation
9B DE F1	203 198-208	ST(1) ← ST/ST(1) pop stack
9B DE F0+i	203 198-208	ST(i) ← ST/ST(i)

FENI
FNENI = Enable Interrupts



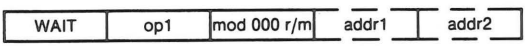
8087 Encoding	Execution Clocks Typical Range	Operation
9B DB E0	5 2-8	clear 8087 interrupt mask
90 DB E0	5 2-8	clear 8087 interrupt mask (no wait)

FFREE = Free Register



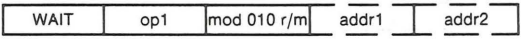
8087 Encoding	Execution Clocks Typical Range	Operation
9B DD C0+i	11 9-16	TAG(i) masked empty

FIADD = Integer Add



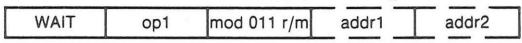
8087 Encoding	Execution Clocks Typical Range	Operation
9B DA m0rm	125+EA (108-143)+EA	ST ← ST + mem-op (short integer)
9B DE m0rm	120+EA (102-137)+EA	ST ← ST + mem-op (word integer)

FICOM = Integer Compare



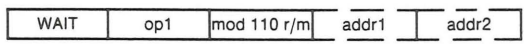
8087 Encoding	Execution Clocks Typical Range	Operation
9B DA m2rm	85+EA (78-91)+EA	ST — mem-op (short integer)
t9B DE m2rm	80+EA (72-86)+EA	ST — mem-op (word integer)

FICOMP = Integer Compare and Pop



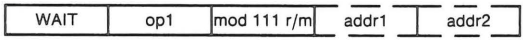
8087 Encoding	Execution Clocks Typical Range	Operation
9B DA m3rm	87+EA (80-93)+EA	ST — mem-op pop stack (short integer)
9B DE m3rm	82+EA (74-88)+EA	ST — mem-op pop stack (word integer)

FIDIV = Integer Divide



8087 Encoding	Execution Clocks Typical Range	Operation
9B DA m6rm	236+EA (230-243)+EA	ST←ST/mem-op (short integer)
9B DE m6rm	230+EA (224-238)+EA	ST←ST/mem-op (word integer)

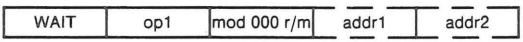
FIDIVR = Integer Divide Reversed



8087 Encoding	Execution Clocks Typical Range	Operation
9B DA m7rm	237+EA (231-245)+EA	ST ← mem-op/ST (short integer)
9B DE m7rm	230+EA (225-239)+EA	ST ← mem-op/ST (word integer)

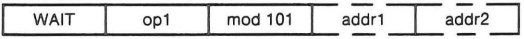
FILD = Integer Load

Word Integer or Short Integer



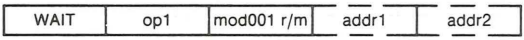
8087 Encoding	Execution Clocks Typical Range	Operation
9B DB m0rm	56+EA (52-60)+EA	push stack ST ← mem-op (short integer)
9B DF m0rm	50+EA (46-54)+EA	push stack ST ← mem-op (word integer)

Long Integer



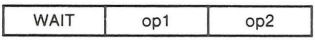
8087 Encoding	Execution Clocks Typical Range	Operation
9B DF m5rm	64+EA (60-68)+EA	push stack ST ← mem-op (long integer)

FIMUL = Integer Multiply



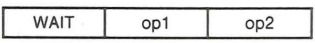
8087 Encoding	Execution Clocks Typical Range	Operation
9B DA m1rm	136 + EA (130-144)+EA	ST ← ST * mem-op (short integer)
9B DE m1rm	130 + EA (124-138)+EA	ST ← ST * mem-op (word integer)

FINCSTP = Increment Stack Pointer



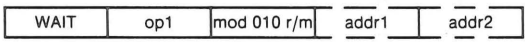
8087 Encoding	Execution Clocks Typical Range	Operation
9B D9 F7	9 6-12	stack pointer ← stack pointer + 1

FINIT / FNINIT = Initialize Processor



8087 Encoding	Execution Clocks Typical Range	Operation
9B DB E3	5 2-8	initialize 8087
90 DB E3	5 2-8	initialize 8087 (no wait)

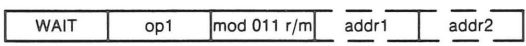
FIST = Integer Store



8087 Encoding	Execution Clocks Typical Range	Operation
9B DB m2rm	88+EA (82-92)+EA	mem-op ← ST (short integer)
9B DF m2rm	86+EA (80-90)+EA	mem-op ← ST (word integer)

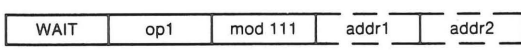
FISTP = Integer Store and Pop

Short Integer or Word Integer



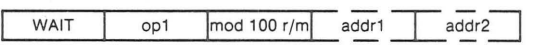
8087 Encoding	Execution Clocks Typical Range	Operation
9B DB m3rm	90+EA (84-94)+EA	mem-op ← ST pop stack (short integer)
9B DF m3rm	88+EA (82-92)+EA	mem-op ← ST pop stack (word integer)

Long Integer



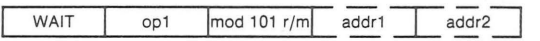
8087 Encoding	Execution Clocks Typical Range	Operation
9B DF m7rm	100+EA (94-105)+EA	mem-op ← ST pop stack (long integer)

FISUB = Integer Subtract



8087 Encoding	Execution Clocks Typical Range	Operation
9B DA m4rm	125+EA (108-143)+EA	ST←ST — mem-op (short integer)
9B DE m4rm	120+EA (102-137)+EA	ST←ST — mem-op (word integer)

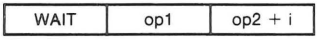
FISUBR = Integer Subtract Reversed



8087 Encoding	Execution Clocks Typical Range	Operation
9B DA m5rm	125+EA (109-144)+EA	ST←mem-op — ST (short integer)
9B DE m5rm	120+EA (103-139)+EA	ST←mem-op — ST (word integer)

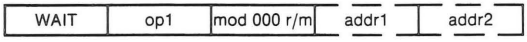
FLD = Load Real

Stack element to Stack top



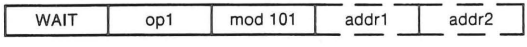
8087 Encoding	Execution Clocks Typical Range	Operation
9B D9 C0+i	20 17-22	$T_1 \leftarrow ST(i)$ push stack $ST \leftarrow T_1$

Memory operand to Stack top
Short Integer or Long Integer



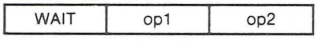
8087 Encoding	Execution Clocks Typical Range	Operation
9B D9 m0rm	43+EA (38-56)+EA	push stack ST ← mem-op (short integer)
9B DD m0rm	46+EA (40-60)+EA	push stack ST ← mem-op (long integer)

Temp Real



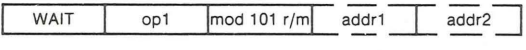
8087 Encoding	Execution Clocks Typical Range	Operation
9B DB m5rm	57+EA (53-65)+EA	push stack ST ← mem-op (temp real)

FLD1 = Load + 1.0



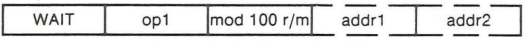
8087 Encoding	Execution Clocks Typical Range	Operation
9B D9 E8	18 15-21	push stack ST ← 1.0

FLDCW = Load Control Word



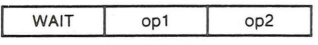
8087 Encoding	Execution Clocks Typical Range	Operation
9B D9 m5rm	10+EA (7-14)+EA	processor control word ← mem-op

FLDENV = Load Environment



8087 Encoding	Execution Clocks Typical Range	Operation
9B D9 m4rm	40+EA (35-45)+EA	8087 environment ← mem-op

FLDL2E = Load Log_2e



8087 Encoding	Execution Clocks Typical Range	Operation
9B D9 EA	18 15-21	push stack ST ← log_2e

FLDL2T = Load $Log_2 10$

WAIT	op1	op2

8087 Encoding	Execution Clocks Typical Range	Operation
9B D9 E9	19 16-22	push stack ST ← $log_2 10$

FLDLG2 = Load $Log_{10} 2$

WAIT	op1	op2

8087 Encoding	Execution Clocks Typical Range	Operation
9B D9 EC	21 18-24	push stack ST ← $log_{10} 2$

FLDPI = Load π

WAIT	op1	op2

8087 Encoding	Execution Clocks Typical Range	Operation
9B D9 EB	19 16-22	push stack ST ← π

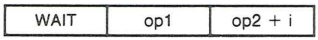
FLDZ = Load + 0.0

WAIT	op1	op2

8087 Encoding	Execution Clocks Typical Range	Operation
9B D9 EE	14 11-17	push stack ST ← 0.0

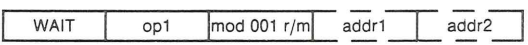
FMUL = Multiply Real

Stack top and Stack element



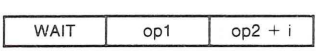
8087 Encoding	Execution Clocks Typical Range	Operation
9B D8 C8+i	138 130-145	ST ← ST * ST(i)
9B DC C8+i	138 130-145	ST(i) ← ST(i) — ST

Stack top and memory operand



8087 Encoding	Execution Clocks Typical Range	Operation
9B D8 m1rm	118+EA (110-125)+EA	ST ← ST * mem-op (short real)
9B DC m1rm	161+EA (154-168)+EA	ST ← ST * mem-op (long real)

FMULP = Multiply Real and Pop



8087 Encoding	Execution Clocks Typical Range	Operation
9B DE C9 +i	142 134-148	ST(i) ← ST(i) * ST pop stack

FNOP = No Operation

WAIT	op1	op2

8087 Encoding	Execution Clocks Typical Range	Operation
9B D9 D0	13 10-16	ST ← ST

FPATAN = Partial Arctangent

WAIT	op1	op2

8087 Encoding	Execution Clocks Typical Range	Operation
9B D9 F3	650 250-800	T_1 ← arctan (ST(1)/ST) pop stack ST ← T_1

FPREM = Partial Remainder

WAIT	op1	op2

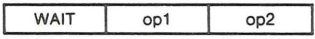
8087 Encoding	Execution Clocks Typical Range	Operation
9B D9 F8	125 15-190	ST ← REPEAT (ST — ST(1))

FPTAN = Partial Tangent

WAIT	op1	op2

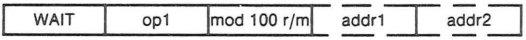
8087 Encoding	Execution Clocks Typical Range	Operation
9B D9 F2	450 30-540	Y/X ← TAN (ST) ST ← Y push stack ST ← X

FRNDINT = Round to Integer



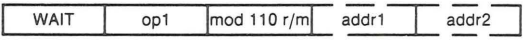
8087 Encoding	Execution Clocks Typical Range	Operation
9B D9 FC	45 16-50	ST ← nearest integer (ST)

FRSTOR = Restore Saved State



8087 Encoding	Execution Clocks Typical Range	Operation
9B DD m4rm	202+EA (197-207)+EA	8087 state ← mem-op

FSAVE
FNSAVE = Save State



8087 Encoding	Execution Clocks Typical Range	Operation
9B DD m6rm	202+EA (197-207)+EA	mem-op ← 8087 state
90 DD m6rm	202+EA (197-207)+EA	mem-op ← 8087 state (no wait)

FSCALE = Scale

WAIT	op1	op2

8087 Encoding	Execution Clocks Typical Range	Operation
9B D9 FD	35 32-38	$ST \leftarrow ST * 2^{[ST(1)]}$

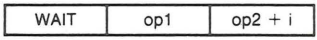
FSQRT = Square Root

WAIT	op1	op2

8087 Encoding	Execution Clocks Typical Range	Operation
9B D9 FA	183 180-186	$ST \leftarrow \sqrt{ST}$

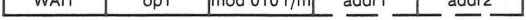
FST = Store Real

Stack top to Stack element



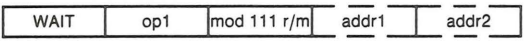
8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DD D0+i	CD 1D D0+i	18 15-22	ST(i) ← ST

Stack top to memory operand



8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 m2rm	CD 19 m2rm	87+EA (84-90)+EA	mem-op ← ST (short-real)
9B DD m2rm	CD 1D m2rm	100+EA (96-104)+EA	mem-op ← ST (long-real)

FSTCW
FNSTCW = Store Control Word



8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 m7rm	CD 19 m7rm	15+EA (12-18)+EA	mem-op ← processor control word
90 D9 m7rm	CD 19 m7rm	15+EA (12-18)+EA	mem-op ← processor control word (no wait)

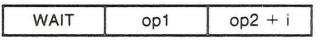
FSTENV
FNSTENV = Store Environment

WAIT	op1	mod 110 r/m	addr1	addr2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 m6rm	CD 19 m6rm	45+EA (40-50)+EA	mem-op ← 8087 environment
90 D9 r6rm	CD 19 m6rm	45+EA (40-50)+EA	mem-op ← 8087 environment (no wait)

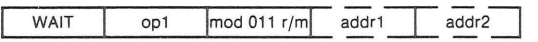
FSTP = Store Real and Pop

Stack top to Stack element



8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DD D8+i	CD 1D D8+i	20 17-24	ST(i) ← ST pop stack

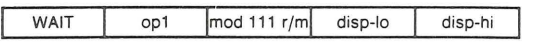
Stack top to memory operand



Long Real or Short Real

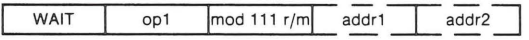
8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 m3rm	CD 19 m3rm	89+EA (86-92)+EA	mem-op ← ST pop stack (short-real)
9B DD m3rm	CD 1B m3rm	102+EA (98-106)+EA	mem-op ← ST pop stack (long-real)

Temp Real



8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DB m7rm	CD 1D m7rm	55+EA (52-58)+EA	mem-op ← ST pop stack (temp-real)

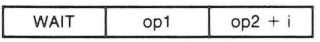
FSTSW
FNSTSW = Store Status Word



8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DD m7rm	CD 1D m7rm	15+EA (12-18)+EA	mem-op ← 8087 status word
90 DD m7rm	CD 1D m7rm	15+EA (12-18)+EA	mem-op ← 8087 status word (no wait)

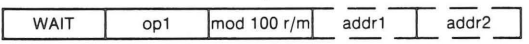
FSUB = Subtract Real

Stack top and Stack element



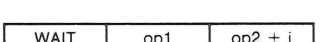
8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D8 E0+i	CD 18 E0+i	85 70-100	ST ← ST — ST(i)
9B DC E8+i	CD 1C E8+i	85 70-100	ST(i) ← ST(i) — ST

Stack top and memory operand



8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D8 m4rm	CD 18 m4rm	105+EA (90-120)+EA	ST ← ST — mem-op (short-real)
9B DC m4rm	CD 1C m4rm	110+EA (95-125)+EA	ST ← ST — mem-op (long-real)

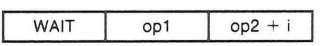
FSUBP = Subtract Real and Pop



8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DE E9	CD 1E E9	90 75-105	ST(1) ← ST(1) — ST pop stack
9B DE E8+i	CD 1E E8+i	90 75-105	ST(i) ← ST(i) — ST pop stack

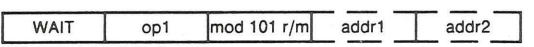
FSUBR = Subtract Real Reversed

Stack top and Stack element



8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D8 E8+i	CD D8 E8+i	87 70-100	ST ← ST(i) — ST
9B DC E0+i	CD 1C E0+i	87 70-100	ST(i) ← ST — ST(i)

Stack top and memory operand



8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D8 m5rm	CD 18 m5rm	105+EA (90-120)+EA	ST ← mem-op — ST (short-real)
9B DC m5rm	CD 1C m5rm	110+EA (95-125)+EA	ST ← mem-op — ST (long-real)

FSUBRP = Subtract Real Reversed and Pop

WAIT	op1	op2 + i

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DE E1	CD 1E E1	90 75-105	ST(1)←ST — ST(1) pop stack
9B DE E0+i	CD 1E E0+i	90 75-105	ST(i)←ST — ST(i) pop stack

FTST = Test Stack Top Against + 0.0

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 E4	CD 19 E4	42 38-48	ST←ST — 0.0

FWAIT = (CPU) Wait While 8087 Is Busy

WAIT

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B	90	3+5n 3+5n	8086 wait instruction

FXAM = Examine Stack Top

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 E5	CD 19 E5	17 12-23	set condition code

FXCH = Exchange Registers

WAIT	op1	op2 + i

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 C8	CD 19 C8	12 10-15	$T_1 \leftarrow ST(1)$ $ST(1) \leftarrow ST$ $ST \leftarrow T_1$
9B D9 C8+i	CD 19 C8+i	12 10-15	$T_1 \leftarrow ST(i)$ $ST(i) \leftarrow ST$ $ST \leftarrow T_1$

FXTRACT = Extract Exponent and Significand

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 F4	CD 19 F4	50 27-55	$T_1 \leftarrow$ exponent (ST) $T_2 \leftarrow$ significand (ST) $ST \leftarrow T_1$ push stack $ST \leftarrow T_2$

FYL2X = Compute Y * Log_2 X

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 F1	CD 19 F1	950 900-1100	$T_1 \leftarrow ST(1) * \log_2 (ST)$ pop stack $ST \leftarrow T_1$

FYL2XP1 = Compute Y * Log_2 (X + 1)

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 F9	CD 19 F9	850 700-1000	$T_1 \leftarrow ST + 1$ $T_2 \leftarrow ST(1) * \log_2 T_1$ pop stack $ST \leftarrow T_2$

NOTES:

NOTES:

NOTES:

NOTES:

NOTES:

UNITED STATES, Intel Corporation
3065 Bowers Ave., Santa Clara, CA 95051
Tel: (408) 987-8080

JAPAN, Intel Japan K.K.
5-6 Tokodai Toyosato-machi, Tsukuba-gun, Ibaraki-ken 300-26
Tel: 029747-8511

FRANCE, Intel Paris
1 Rue Edison, BP 303, 78054 Saint-Quentin-en-Yvelines Cedex
Tel: (33) 1-30-57-7000

UNITED KINGDOM, Intel Corporation (U.K.) Ltd.
Pipers Way, Swindon, Wiltshire, England SN3 1RJ
Tel: (0793) 696000

WEST GERMANY, Intel Semiconductor GmbH
Seidlestrasse 27, D-8000 Muenchen 2
Tel: (89) 53891

HONG KONG, Intel Semiconductor Ltd.
1701-3 Connaught Centre, 1 Connaught Road
Tel: (5) 844-4555

Printed in U.S.A./TP119/3K/PE SM
Embedded Controllers